System for Rapid Subtitling

by

Sean Joseph Leonard

Submitted to the Department of Electrical Engineering and Computer Science

in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 12, 2005

Author.....
Department of Electrical Engineering and Computer Science
September 12, 2005

Certified by.....
Harold Abelson
Class of 1922 Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by.....
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

System for Rapid Subtitling

by

Sean Joseph Leonard

Submitted to the Department of Electrical Engineering and Computer Science

on September 12, 2005, in partial fulfillment of the

requirements for the degree of

Master of Engineering

Abstract

A system for rapid subtitling of audiovisual sequences was developed, and evaluated. This new system resulted in average time-savings of 50% over the previous work in the field. To subtitle a 27-minute English lecture, users saved 2.2 hours, spending an average of 1.9 hours timing the monologue. Subtitling a 20-minute Japanese animation resulted in similar savings.

While subtitling is widely practiced, traditional speech boundary detection algorithms have proven insufficient for adoption when subtitling audiovisual sequences. Human-edited timing of subtitles is a dull and laborious exercise, yet with some human-guidance, basic systems may accurately align transcripts and translations in the presence of divergent speakers, multiple languages, and background noise.

Since no known, fully-automatic recognition systems operate with sufficient accuracy for production-grade subtitle times, during playback the system disclosed treats user-supplied times as *a priori* data and adjusts those times using extracted features from concurrent data streams. In the application's oracle subsystem, packaged algorithms called oracles preprocess, filter, and present media data, using that data to adjust user-supplied times. The on-the-fly timing subsystem gathers user-supplied times and coordinates the oracles' adjustment of the data. The application was developed with attention to user interface design and data format support, and was sent to users for structured feedback. From these data, system performance is discussed in terms of subtitle time accuracy, time spent by users, and avoidance of common pitfalls in subtitle generation.

Thesis Supervisor: Harold Abelson
Title: Class of 1922 Professor of Computer Science and Engineering

Acknowledgements

There are many people to whom I want to express my undying gratitude. To my parents, who have supported me emotionally for all these years and particularly for my final summer at MIT. Thanks to my advisor, Hal Abelson, who has supported this thesis and continuously offered technical advice as well as healthy doses of project management reality. Thanks especially to all of the alpha and beta testers, who provided their valuable time and feedback.

Contents

List of Figures

1. Introduction

This thesis presents a system for rapid subtitling of audiovisual sequences. This new system resulted in significant time-savings for users timing two different audiovisual streams. Rather than a reported average of 4.1 hours (normalized)[1] using previous tools, testers spent an average of 1.9 hours timing an English-language lecture. Subtitling a 20-minute Japanese animation sequence resulted in similar savings: testers with exposure to Japanese animation spent an average of 1.5 hours instead of 3.7 hours with previous tools. However, these savings were distributed unevenly across the different timing stages, suggesting that more research and testing is necessary, using the current system as a reference point for further development.

1.1. Overview of Results, Motivation, Methodology

Subtitling is important to academia and industry in the global economy. My interest in this field stems from my interest in Japanese animation, where subtitling of Japanese-language works is a regular practice. I developed a subtitling system, codenamed SubTek™,[2] that streamlines the preparation of subtitles. I established a user base for SubTek

[1] In the preliminary survey, users reported how long it took to time a 22-minute episode in the target language, but the sample lecture is 27 minutes long. This value has been scaled by a factor of 27/22.

[2] Alternate names are under consideration when this software is released to the public. The current proposal is Foresight.

and evaluated the software's usefulness in four key user groups using qualitative and quantitative metrics. In this thesis, I will discuss the system's implications for subtitling efforts and thus for broader concerns about the globalization and dissemination of culture.

To achieve the overall time-savings reported above, SubTek—the System for Rapid Subtitling—addresses three problem domains: timing, user interface, and format conversion. The system makes both technical and "build a cool tool for the world" contributions building on previous work in the field. SubTek implements a novel framework for *timing* subtitles, or specifying when a subtitle appears and disappears onscreen.

The On-the-Fly-Timing Subsystem, described in System Design (p. 16), uses parameters derived from the subtitle, audio, and video streams, in combination with user input, to rapidly produce and assign accurate subtitle times. With SubTek, subtitlers can typeset their work to enhance the readability and visual appearance of text onscreen. Subtitlers may prepare and process subtitles in many formats using SubTek's modular serialization framework.

As described in System Evaluation (p. 43), I consulted four distinct categories of users during software design, development, and testing: professionals, academics, fans, and novice users. Each group expressed particular concerns about the design of its ideal subtitling package. For

instance, fan subtitlers were especially concerned about typesetting and animation capabilities, while typesetting features are of secondary importance to data and time format support among subtitling professionals. My software addresses the peculiarities of subtitling in the Japanese animation community, but easily generalizes to the subtitling of media in other languages. These concerns influenced the development and feature set of SubTek, hopefully leading to an implementation that balances many user groups' interests without adding needless complexity via from features added on top of and in conflict with the central design.

SubTek, with its clean user interface, advanced timing system, and wide support for file formats, enabled users to reduce their time spent subtitling a 27 minute lecture in English from 4.1 hours to 1.9 hours. System Evaluation (p. 43), explores the derivation and computation of these values from the user feedback provided. While this time decrease is very promising, this research suggests several directions where future work needs to be done, described in Feedback and Future Directions (p. 64) and Analysis and Conclusion (p. 68).

2. Historical Work

2.1. Historical Background

There is a need for a subtitling program such as SubTek because while "modern" subtitlers exist for the aforementioned users, none address the core

problem of labor efficiency during the timing process. The few commercial subtitling systems in existence have small and exclusive user bases, primarily consisting of large broadcasting houses. Their cost and complexity are beyond the reach of fans, academics, and freelance translators; at the same time, at least one broadcast industry source has claimed that these commercial systems are even less stable than their open-source and freeware counterparts.

Furthermore, no known systems fully implement "i18n" (internationalization) features such as Unicode, language selection, collaborative translation, multilingual font selection, or scrolling text. The plethora of subtitling software has led to hundreds of different file formats for subtitle text. Adding an additional format to this list would only exacerbate the problem.

2.2. Sub Station Alpha

Among those who subtitle on a regular basis, only one software system consistently appears in the literature: Sub Station Alpha 4.08 (SSA). SSA was released as freeware by Kotus, an individual developer, but the software has not been updated since 2000 and the source code was never released. SSA is based on workflows for hardware character generator-locking devices (*genlocks*), a technology eclipsed nearly half a decade ago by all-digital

workflows. According to one professional translator, subtitling a 25-minute video sequence requires about 4 hours with current tools.

2.3. The SSA Legacy

Consider in Figure 1 the user interface for Sub Station Alpha, perhaps the most popular program for non-professional subtitling:

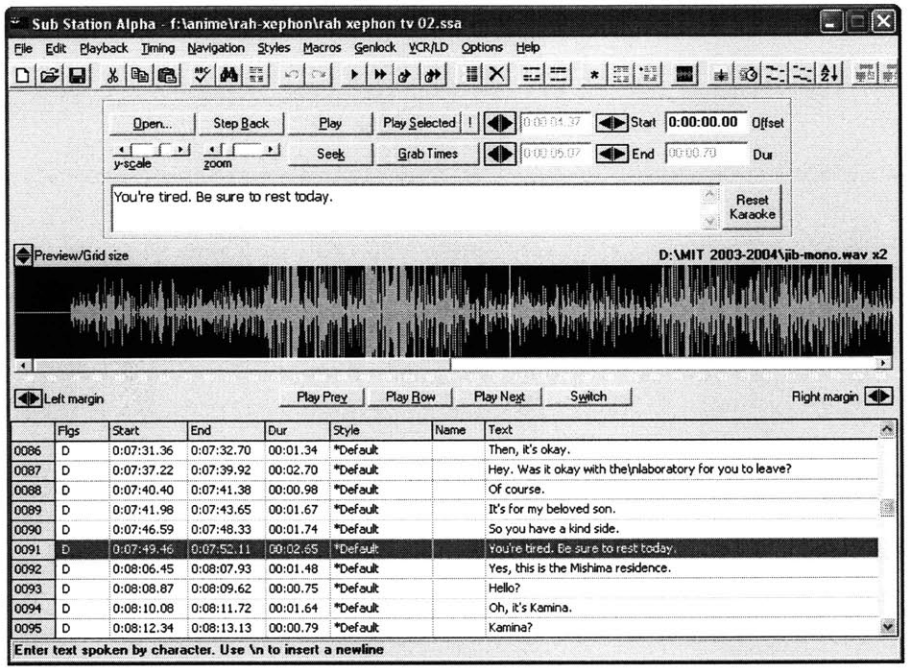


Figure 1: Sub Station Alpha 4.08. The "linear timeline view" limits a user's ability to distinguish concurrent subtitle data.

Sub Station Alpha's "linear timeline view" is straightforward in its implementation, but suffers from several drawbacks. First, the "Preview/Grid size" area serves as both the preview window for subtitles and the audio waveform, so it is not possible to see all of a subtitle while editing. Keyboard shortcuts are awkward or nonfunctional. The waveform preview acts inconsistently: sometimes a click will update the time, other times it will not.

Fourth, subtitles are arranged in single-file order down the table (bottom); there is no attempt to organize or filter subtitles by author, character, or style, and there is no option to view multiple subtitle sections at once.

In spite of these problems, Sub Station Alpha is a mature solution that has stood the test of time. It almost never crashes during daily use, and its centisecond-based timing format, though awkward, provides enough resolution to time both digital and analog formats with intermediate conversion programs. Although not shown here, Sub Station Alpha implements a live preview mode where the text is rendered against a solid background using Microsoft® DirectDraw technology, usable for chroma-key overlay applications. Some professionals have used this mode to time subtitles on-the-fly, but with limited success for initial passes through the data.

2.4. Sub Station Alpha Successors

The subtitling field is rife with several projects: among them include freeware applications like Sabbu [1] (Figure 2 and Figure 3, below), Medusa [2], and XombieSub [3]. "Professional-class" applications such as Lemony [4] exist, as well as applications such as SoftNI [5], which is targeted at broadcast studios. Although some of these programs predate Sub Station Alpha, their feature sets vary considerably, with different approaches to layouts, emphasis on multilingual support, or video preview windows. These

tools tend to support a large handful—possibly a dozen or more—file formats, but many of these tools create their own proprietary formats as well.

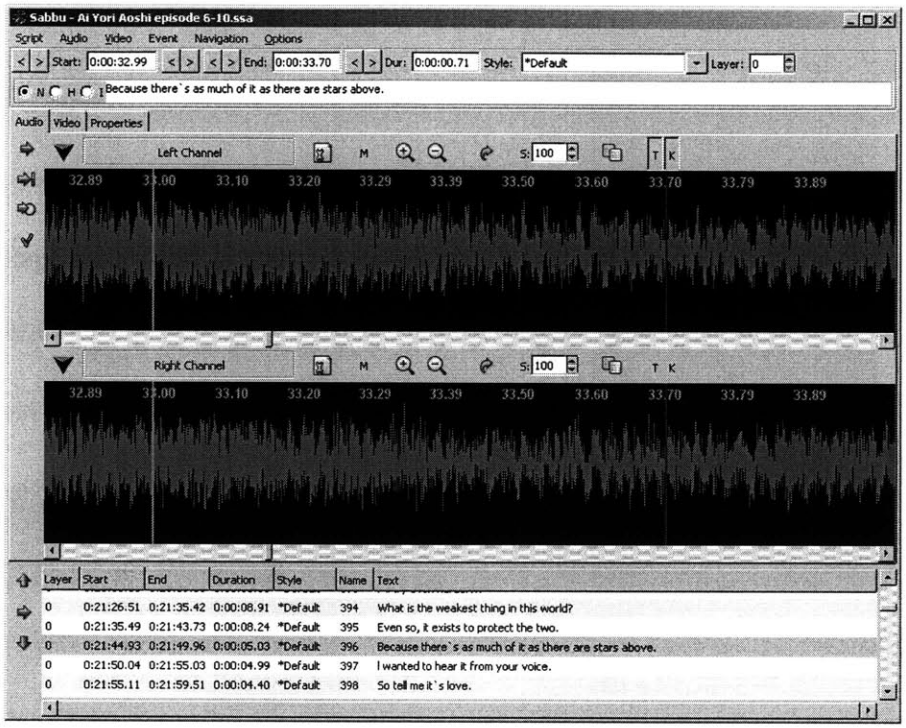


Figure 2: Sabbu 0.1.5 (December 2004). An improvement of Sub Station Alpha, but fundamentally the same interface.

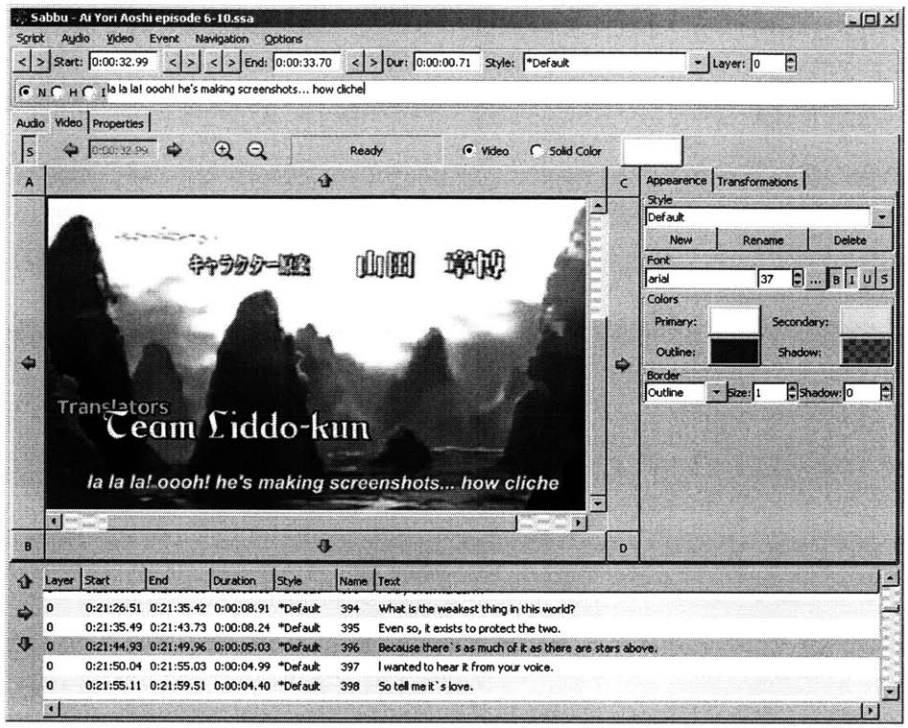


Figure 3: Sabbu 0.1.5 Video Preview, but not real time video rendering.

Despite these variations, the user interfaces of these programs bear many similarities either to nonlinear editing tools such as Adobe Premiere, or grid-and-preview-model systems such as Sub Station Alpha itself. Perhaps most significantly, in these programs the video window is considered an adjunct, child, or popup window to the script window; the script window is almost always a single-line grid or a nonlinear editing track.

2.5. Framing the Approach

To improve on existing interfaces, designing SubTek required completely reevaluating the needs of subtitlers, while recognizing that without continuity, many users of previous tools would feel frustrated. I began by interviewing current practitioners as well as complete novices,

gathering their recommendations and sketching sample user interfaces. SubTek's current user interface is detailed in System Design (p. 16) and the discussion of its development follows in System Evaluation (p. 43).

2.6. Review of Automated Speech and Scene Detection Methods

The problems raised by automation in my subtitling system are domains with substantial research literature. Although the current iteration of SubTek does not implement a sophisticated audio recognition module, it supports infrastructure for it, and has been designed with support for interchangeable feature detection modules based on future work. Therefore, reviewing literature on feature classification for speech and scene boundary detection remains worthwhile when considering what SubTek can become, as well as what SubTek already does well. Combining a foreign language transcript and a audiovisual sequence into a subtitled work raises two distinct problem domains: speech boundary detection and phonetic audio alignment, and video scene boundary and character recognition.

A sizeable corpus of research has been conducted on speech recognition and synthesis. Phonetic alignment falls under this broad category, and multiple systems exist in fully-specified paper form. For example, [6] implemented a SPHINX-II system that is available for implementation. Recent work by [7] suggests that my subtitling system is possible to implement for cases when the repertoire of the recognition system is limited.

Japanese language has many notable complications. Most systems for phonetic alignment have been tested in limited English, not in large-volume Japanese or other languages. Further research is necessary. It is possible that the repertoire of syllables be fewer than English (Japanese has fewer *mora*, or syllable-units, than English), but Japanese tends to be spoken faster than English, and the phonetic alignment routine must treat a complex and noisy waveform. In literature on the topic, researchers almost always provide a single, unobstructed speaker as input data to their systems. Using an audio stream that includes music, sound effects, and other speakers presents additional algorithmic challenges that require additional study.

Likewise, Japanese animation tends to cast a great variety of characters with a few voice-types. Small variations between speakers may confuse the alignment routine, and may prevent detection of speaker change when two similar voices are talking in succession (or worse yet, at the same time). Fortunately, however, transcripts and translations in the subtitling sphere come pre-labeled with character names. Since characters are known *a priori*, one might consider operating speech signature detection "in cooperative:" given known, well-timed subtitles, a classification algorithm can extract audio data from these known samples, and determine which areas of the unknown region correspond to the character, to another character, or to no character at all.

Recent work by [8] discuss video scene boundary detection based on the principle of continuity, but their results have only 82.7% precision after an intensive study of footage. While impressive for videos, 82.7% precision is far too low for freestanding subtitle implementations.

3. System Design

At 30,129 net source lines of code, SubTek beta 1 may seem formidable and complex. Therefore, this discussion of system design begins with a summary of its key user interface elements, progressively revealing complexity in a series of figures. The On-the-Fly Timing Subsystem and Oracle Subsystem are described next, as these systems form the core technical advance of SubTek. The remaining five system feature groups are then discussed: choice of platform, user interface of the script and video views, data storage, manipulations, and consistency of Observer design pattern [9], internationalization via Unicode, and localization via resource tagging.

3.1. Overall Design

The SubTek Application object is a singleton that forms the basis for execution and data control. The application creates and holds references to the Script frame and its views (collectively hereinafter "Script View"), and the Video & Oracles frame and view (collectively hereinafter "Video View"), as shown in Figure 4. Unlike most previous subtitling applications, which may

put video or media presentation in a supporting role to the script, both Script View and Video View are equally important in SubTek. Both views are full windows with distinct user interfaces. The user can position these views anywhere and on any monitor he or she feels comfortable with.

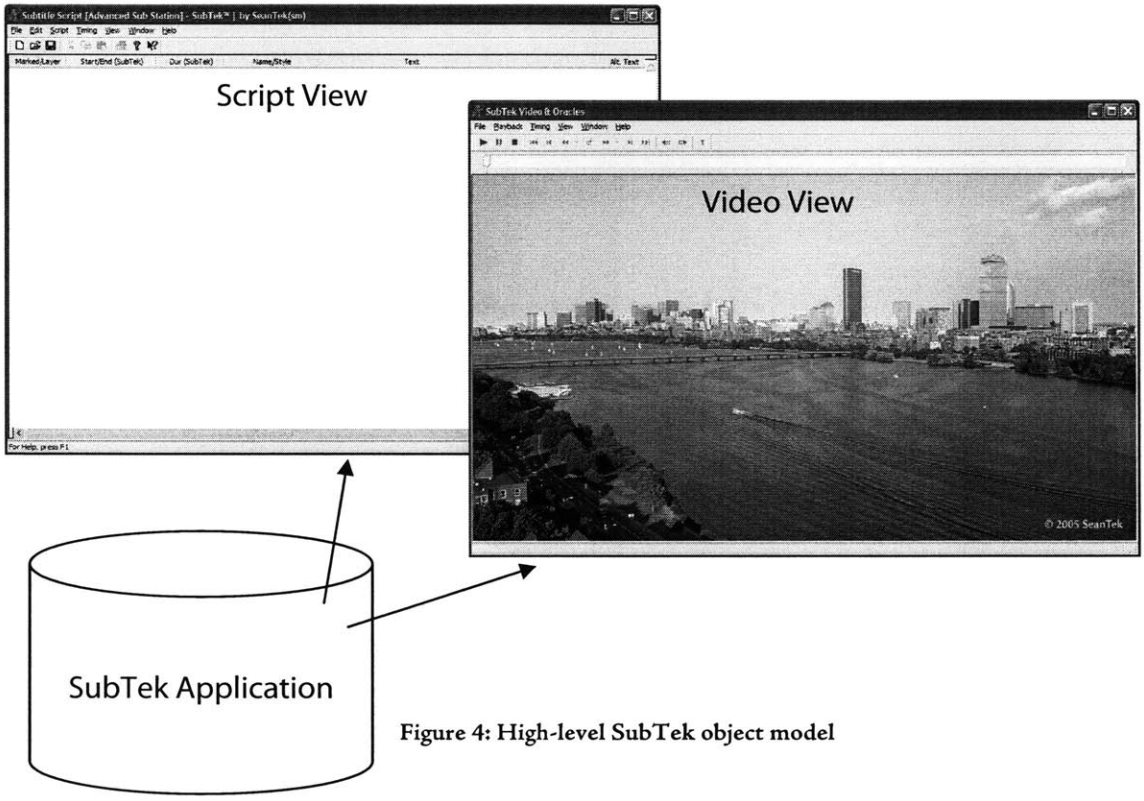


Figure 4: High-level SubTek object model

When SubTek is launched, the application object loads, performs initialization of objects, and reads saved preferences from the system and the system preference store. Then, the application object loads Script View and Video View. From Script View, users interact directly with the subtitles and data in the script, including loading scripts from and saving scripts to disk via serialization objects. A distinct Script object holds the script's data; all modules communicate the Script object.

In Video View, users load media clips and play them back using a filter graph manager[3] and customized filters. In this context, *filters* are sources, transforms, or renderers. Data is pushed through a series of connected filters from sources to renderers; the renderers in turn deliver media data to hardware, *i.e.*, the audio and video cards. SubTek provides a "preview filter" mechanism that renders formatted subtitles atop the video stream. Since the focus of this thesis is on timing rather than typesetting, however, the current version of SubTek does not come bundled with a formatted subtitle display module. If such a display module is present on the user's system, SubTek will attempt to load that module for presenting real-time subtitles. At the end of the video chain, a highly customized video renderer[4] uses 3D acceleration on the graphics card to prepare and present video.

---

[3] Utilizing the Microsoft® DirectShow API.

[4] The module is called the Video Mixing Renderer 9 Custom Allocator-Presenter, or VMRAP9.

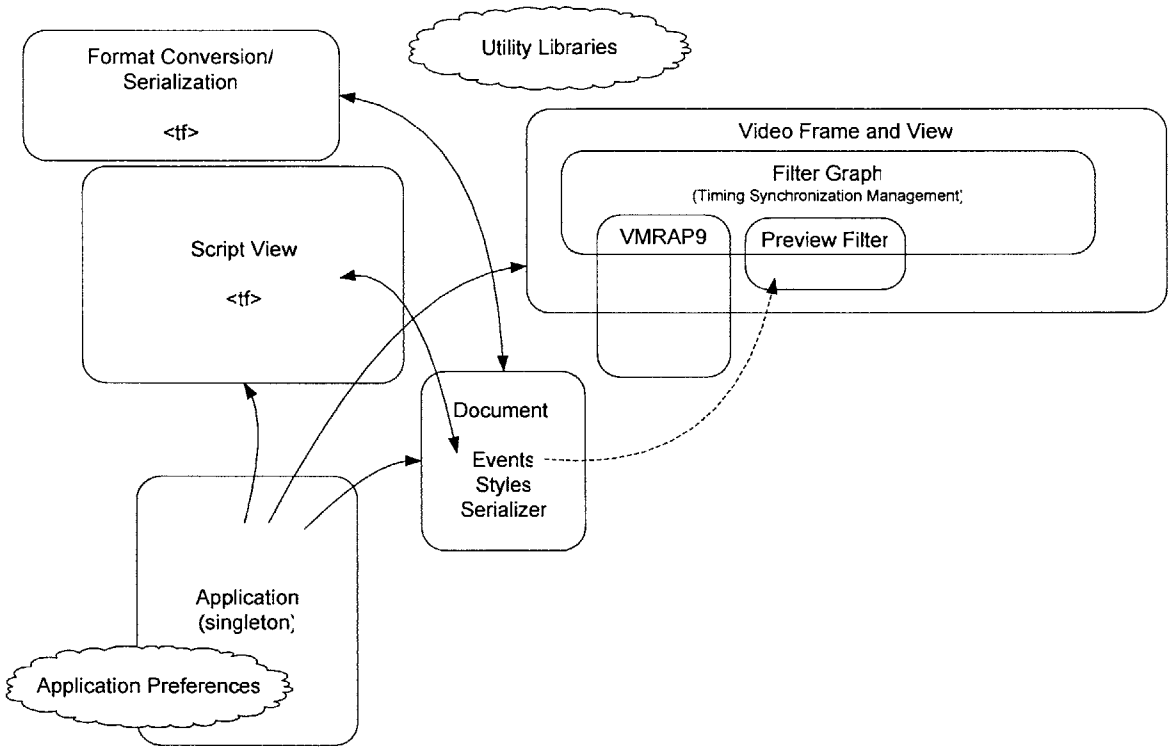


Figure 5: Partial SubTek object model, with objects, data flows, and observation cycles as described in text.

Figure 5 shows the aforementioned objects as well as application preferences, utility libraries, and transform filters. Rounded rectangles are objects; overlapping objects indicate "owner-owned" relationships. Single-headed arrows indicate awareness and manipulation of the pointed-to object by the pointing object: the Application, for example, creates and destroys the Script and Video Views in response to system events. The single-headed dotted-line arrow indicates an observer-subject relationship: the preview filter receives updates when events in the Script object change. Double-headed arrows indicate mutual dependencies between two objects or systems. Modules throughout the system use application preferences and utility libraries, so specific connections are not shown; rather, these objects are

indicated as clouds. In this context, *transform filters* are first-class function objects, or closures, that transform Script object elements and filter them into element subsets. Transform filters appear in Figure 5 as <tf>. A thorough discussion of transform filters follows in Data Storage, Transformation, and Consistency (p. 37).

Figure 6 completes the SubTek object model with the On-the-Fly Timing Subsystem and Oracle Subsystem, as described in the following section.

3.2. On-the Fly Timing Subsystem and Oracle Subsystem

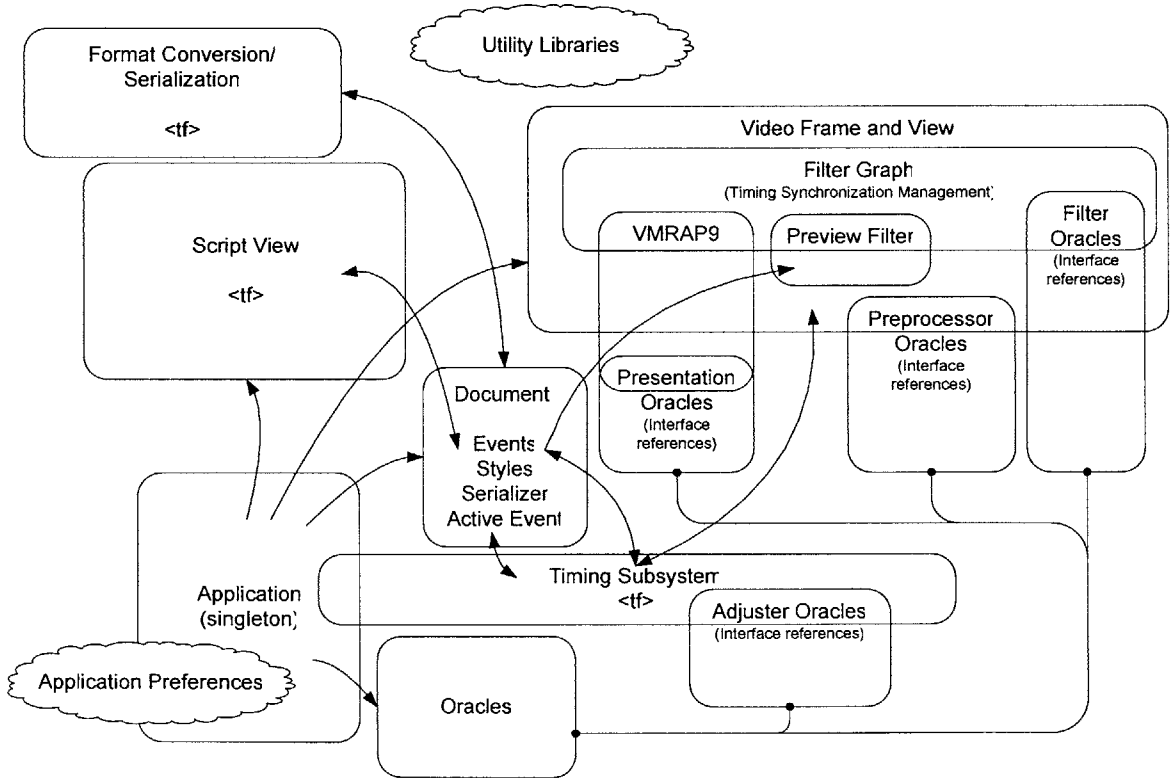


Figure 6: Complete SubTek object model, including the On-The-Fly Timing Subsystem and Oracle Subsystem. Circle-headed connectors indicate single objects exposing multiple interfaces to different client objects.

The most important technical advance in this thesis work is the design of the On-the-Fly-Timing Subsystem and Oracle Subsystem. These subsystems control and automate the selection of event start and end times. As discussed above in Review of Automated Speech and S (p. 14), even the most sophisticated video and audio processing algorithms alone cannot reach the levels of accuracy required in the subtitling process. In particular, speech boundary detection algorithms tend to generate far too many false positives due to breaks in speech or changes to temp for dramatic effect. Even if an automated process can track audiovisual cues with 100% accuracy, a human still must confirm that generated times are optimal by watching the audiovisual sequence before audiences do. Audiences expect subtitles not to merely track spoken dialogue, but to express the artistic vision of the film or episode. Just as a literal translation would do violence to the narrative, so too would mechanical tracking destroy the suspense, release, and enlightenment of the "visual dialogue." This constraint differs from live captioning of television broadcasts such as news and sports, where the content is so fresh that temporary desynchronization is acceptable because the objective is receipt of raw, undecorated information.

Since human intervention is inevitable, SubTek treats user-supplied times as *a priori* data and adjusts these inputs based on packaged algorithms that extract features from concurrent data streams or from the user's

preferences. "User-supplied times" may be provided by any process external to the two subsystems. In another implementation, times may be "batched up," saved to disk, and replayed or provided in one large, single adjust request. A more complete discussion of this alternative follows at the end of Timing Subsystem Operation (p. 27).

Algorithms are packaged in Oracle objects, which expose one or more oracle interfaces: Preprocessor, Filter, Presenter, and Adjuster, according to the Interface Segregation Principle [10,11]. The Application object distributes ordered lists of these interface references to appropriate subsystems. These subsystems invoke appropriate commands on the interfaces, in the order provided by the Application objects.

While discussing these interfaces, consider one such oracle as an example, the Video Keyframe Oracle. Invoking the Preprocess method on the Preprocessor interface causes an oracle to preprocess the newly-loaded file or remove the newly-unloaded file. The video keyframe oracle preprocesses the stream by opening the file, scanning through the entire file, and adding key frames to a set sorted by frame start time. As a performance optimization, the video keyframe oracle's Preprocess launches a worker thread that scans the file using a private filter graph while the video view continues to load and play.

The Filter interface is similar to Preprocess, in that one of its objectives may be to analyze stream data. However, another possible scenario is to transform data passing through the Video View's filter graph in response to events on one of the other interfaces. The primary constraint of a media filter is that it cannot manipulate the filter graph directly, so it cannot, for example, pre-fill large buffers with data ahead of the current media time.

The Presenter interface is invoked before the 3D rendering back buffer is copied to screen. While SubTek provides a predefined area of the screen to update, the oracle may draw to any point in 3D space. The video keyframe oracle uses presentation time information to render the keyframes as lines on a scrolling display. Oracles are multithreaded objects, so great care was taken to synchronize access to shared variables while preventing deadlocks.

Adjuster is the most complicated interface, as the On-the-Fly Timing Subsystem uses this interface to notify oracles of user-generated events and to adjust times in the *oracle pipeline*. Examining the role of the Adjuster interface from the other subsystem is more instructive, as the pseudocode for On-the-Fly Timing demonstrates in Appendix B. Since the timing subsystem first compiles user-generated events into a structure for the oracle pipeline, I will first review the possible final-subtitle scenarios to build a case for the timing system's behavior.

3.2.1. Subtitle Transition Scenarios

Since events during on-the-fly timing pass in real time, the user has very little chance to react by issuing many distinct signals, *i.e.,* by pressing many distinct keys, when a subtitle is to begin or end. There are eight possible transitions between subtitles; our objective is to map signals to scenarios while reducing or eliminating as many scenarios as possible. Each scenario contains a mini-timeline: speakers of subtitles are "characters" named A, B, etc., while the specific subtitle for that character is listed by number. Empty space indicates no one is speaking at that time. The right arrow indicates that time is increasing towards the right.

1. *Characters speaking individually and distinctly.* A1 B1 A2 B2 t →

   This is the basic scenario requiring one signal pair: start (transition to signal) and end (transition to non-signal), corresponding to the start and end times of an event.

2. *A character speaking individually but not distinctly.* A1A2A3 t →

   Characters may speak a prolonged monologue that cannot be displayed naturally as one subtitle. Assuming that it is not possible to issues two signals at th exact same time, there must be three signals: "start," "adjacent," and "end."

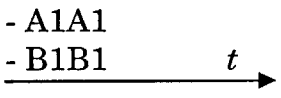
3. *A character speaking individually but not very* A1.A2.A3 t →

   distinctly. This scenario is similar to scenario 2, except that it may or may not be possible to issue two separate sets of signals given human

reaction time. Speakers temporarily stopping at a "natural pause" would fit this scenario. If this scenario is treated as 2, the adjustment phase, rather than the human signaling phase, must distinguish between these times.

4. *Characters speaking indistinctly.* In a heated dialogue between two or more speakers, it may not be possible to signal distinct start and end times. However, we know who is speaking (character A or B) from the translated or transcripted dialogue, which lists the speaker. This *a priori* knowledge may serve as a strong hint to the adjustment phase; for the human signaling phase, this knowledge means that the signals need not be distinct. Therefore, scenario reduces to scenario 2.

5. *Characters speaking in a dialogue on the same subtitle (typically delimited by hyphens at the beginning of lines).* While it is unlikely that multiple characters will speak the exact same utterances at the exact same times, the combination of events in the subtitle data reduces this scenario to scenario 1, with one signal pair. It is more likely, however, that a human operator will err by issuing false positives at the actual transition in speech: A stops talking and B starts talking, but the



human operator fails to see that A and B talking are in the same event. Therefore, a "go back" signal may be desired.

6. *Non-character with subtitle.* A translator's note or informational point may appear onscreen while a character is talking. Typically, however, these collisions occur only temporally, not spatially: the translator's note may be rendered as a supertitle. In this case, the human operator may generate either no signal or an "ignore" signal; the best approach, however, is to filter out non-character events so that they are not presented during timing.

7. *Collisions: characters interrupt one another.* If this scenario occurs, it occurs very briefly but causes great disruption: A typically stops talking within milliseconds of B starting. While sophisticated processing during the adjustment phase may identify this scenario, preserving the collision is undesirable for technical and artistic reasons. Professional translator Neil Nadelman, for example, reports that many DVD players may crash or otherwise fail when presented with subpicture collisions. Treating scenario 7 as an adjacency, scenario 4, would be technically incorrect from the standpoint of recognition, but practically correct from the standpoint of external technical constraints. On the artistic side, some subtitling professionals report that audiences find collisions jarring, perhaps

more so than the interruption onscreen (if the subtitles collide, the viewer is interrupted in addition to watching the interruption). A translator or transcriptionist would thus reduce this scenario to scenario 5.

8. *Characters utter unsubtitled grunts or other "false-positives" before speaking.* The greatest danger is that the false-positive will lead to a human operator false-positive. However, the error is that the signal is issued too early, rather than too late. This scenario may be addressed by a "restart" signal.

From studying these eight scenarios, we are left with three core signals: "start," "adjacent," and "end," with optional signals meaning "back," "restart," and "next."

3.2.2. Timing Subsystem Operation

While timing mode is active, user-generated events are forwarded to the *SignalTiming* function. *SignalTiming* builds a temporary queue of "adjacent events," then submits the queue for adjustment in the oracle pipeline. In more concrete terms, the Script object stores a reference to the *active event* (a subtitle). When the user depresses the "J" or "K" keys,[5] the timing subsystem stores the time and event. When the key is released, the

[5] The actual keys are customizable, but the keys described in this document are the defaults. These keys correspond to the most natural position in which the right-hand may rest while on a QWERTY keyboard.

time is recorded as the end time, and the queue is sent to the oracle adjustment phase, as described below.

If "J" or "K" is depressed while the other is depressed, *SignalTiming* will interpret this signal as an "adjacent." The time is recorded as the adjacent time corresponding to the end of the active event and the start of the next event, which is designated the new active event. Release of one of these keys will be ignored, but release of the final key results in an "end" signal as above.

Two navigational keys specify "designate the previous event active, and cancel any stored queue without running adjustments" (defaults to "L") and "designate the next event active, canceling the queue" (defaults to ";"). Advanced and well-coordinated users may use "H" to "repeat," or set the previous event active and signal "begin." They may also use "N" to re-signal "begin" on the current active event. Given the difficulty of memorizing one keystroke, however, it is expected that users will use "J" and "K" for almost all of their interactions with the program.

When "end" is signaled, the event queue is considered ready for oracle adjustment. SubTek prepares a two-dimensional array of *pipeline storage elements*; the array size corresponds to the number of stages—equal to the number of Adjuster interfaces—by the number of events plus one. The extra size on the event extent is for processing the end time. Each pipeline storage

element stores primary times, the standard deviation for those times, alternate times, confidence ratings on the alternate times, and a window specifying the absolute minimum and maximum times to search in (Figure 7; *see* Appendix B). While each pipeline segment corresponds to one event and one time (start, adjacent, or end), oracles may separate an adjacent time into unequal "last end" and "next start" times. The oracle for each stage examines the pipeline storage with respect to the current event and stage. The oracle is provided with the best known times from the previous stage, but the oracle also has read access to all events in the pipeline—all previous stages before it are filled with cached times. Storage of and access to this past data is useful, for example, when computing optimal subtitle duration: the absolute time for the current stage depends on the optimal times from previous stages.

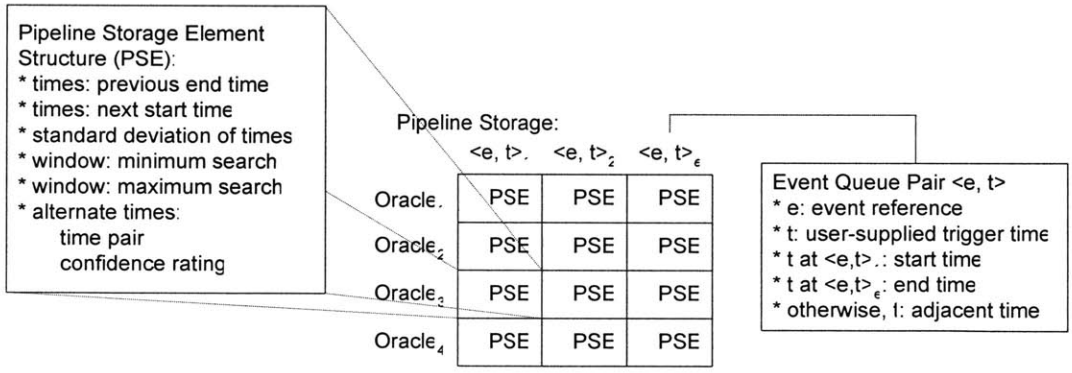


Figure 7: Pipeline storage 2D array and control flow through the pipeline stages.

Pipeline storage further exposes to the oracle subsystem the interfaces of the oracles corresponding to each stage. Each Adjuster interface further exposes a unique identifier of the concrete class or object, so an Adjuster can determine what actually executed before it or what will execute after it.

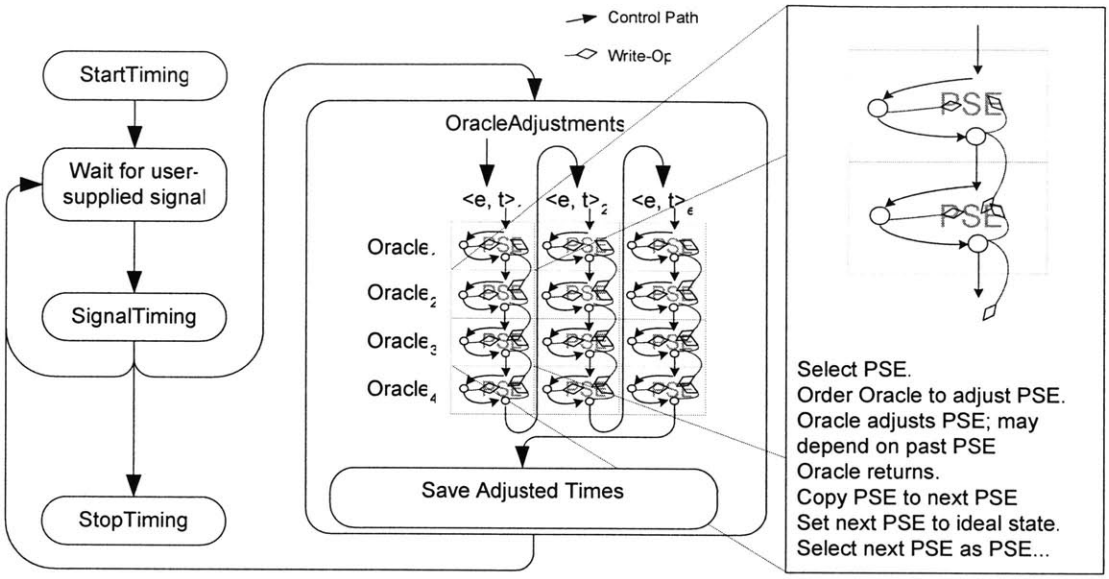


Figure 8: Operations and interactions between On-the-Fly Timing Subsystem and Oracle Subsystem during oracle adjustments. Control paths are indicated by solid lines and arrows; diamond-ended dotted lines show write operations on individual elements of the pipeline.

Control weaves between the on-the-fly timing subsystem and the Adjuster code in the oracle subsystem, as suggested in Figure 8. The Adjust method of the Adjuster interface receives a non-constant reference to its pipeline storage element, into which it writes results. When control passes back to the on-the-fly-timing subsystem, the subsystem may, at its option, adjust or replace the results from the previous Adjuster. At the end of a pipeline segment for an event, the timing subsystem replaces the times of the event with the final-adjusted times.

In principle, these exposures violate the Dependency Inversion Principle of object-oriented programming [10,12], which states that details should depend upon abstractions. However, it is best to think of the oracle adjustment phase as a practically-controlled, rather than formally-controlled,

network of dependencies. The primary control path through the pipeline constitutes normal execution, but a highly-customized mix of oracles may demand custom code and unforeseen dependencies. In this case, a single programmer or organization might create or assemble all of the oracles; such a creator would understand all of the oracles' state dependencies. An advanced user, in contrast, could specify which oracles operate in a particular order in the pipeline for specific behavior, but those effects would be less predictable if one oracle depends on the internal details of another. Finally, if an audio processing algorithm is known to provide spurious results on particular data, a subsequent oracle could test for that particular data and ignore the previous stage's results. Replacing one algorithm with another is as simple as replacing a single oracle interface reference, thus placing emphasis on the whole framework for delivery of optimal times.

Human interaction plays an important role in this framework, but there are alternative modes of operation that one may explore. It is simple to operate this framework outside of real time playback by supplying prerecorded user data or by generating data from another process. There is no explicit requirement that times must strictly increase, for example: the controlling system may generate times in reverse. The filter and presenter interfaces do not have to be supplied to the VMRAP9 and filter graph modules, thus saving processor cycles.

Nevertheless, SubTek beta 1 does not implement these alternatives in light of the aforementioned constraints of the problem domain. First, irrespective of the Interface Segregation Principle, an oracle may use its Presenter or Filter behavior to influence behavior on the other interfaces. Causal audio oracles, for example, might implement audio processing and feature extraction on their Filter interfaces, while a video oracle might read bits from the presentation surface to influence how it will adjust future times passed to it. As will be discussed in Oracles in Use and Pipeline Order (p. 33), the Sub Dur Oracle presents a visual estimate of the duration of the "hot" subtitle, which may subtly influence a user's response. Presenter and Filter interfaces should be seen as part of a larger feedback loop that involves, informs, and stimulates the user.

Second, oracles may save computation time by relying on user feedback from the Adjuster interface to influence data gathering or processing on the other interfaces. A "signage movement detector," for example, might perform (or batch on a low-priority thread) extensive computations on a scene, but only on those scenes where the user has indicated that a sign is currently being watched.[6] Third, it is faster for a user to react in real time to a subtitle, and for a computation to perform an exhaustive search in a limited range,

[6] Although not part of each PSE in the SubTek beta 1 implementation, an oracle can conceivably manipulate events during time-gathering phase, or record events for manipulation in the oracle adjustment phase.

than it is for a computation to search a much more expansive range and require the user to pick from many suboptimal results. To "reverse" operation, the timing subsystem could generate signals in small, equally-spaced intervals and see where those input times cluster after being adjusted by "stateless" oracles. However, the computer is not good at picking from wide ranges of data; humans are not good at quickly identifying precise thresholds. If the user takes care of the macro-identification, the system should take care of the rest.

3.2.3. Oracles in Use and Pipeline Order

I constructed the following oracles for use with the beta version of SubTek. The order presented below corresponds to the order of these oracles in the oracle pipeline:

Sub Queue Oracle (Presenter, Adjuster): Displays the active event and any number of events before ("prev events") and after ("next events") the active event. Since this oracle presents text over the video using Direct3D, it is extremely fast. This oracle does not perform adjustments in the pipeline.

Audio Oracle (Preprocessor, Presenter, Adjuster): Preprocesses audio waveforms by constructing a private filter graph based on the Video View filter graph and continuously reading data from the graph through a sink (a special renderer) that delivers data to a massive circular buffer. The oracle presents the waveform as a 3D object rendered to the presentation area of the

video view, with the vertical extent zoomed to see peaks more easily. The oracle computes the time-based energy of the combined-channel signal using Parseval's relation [13] and a windowing function. The oracle adjusts the event time by picking the sharpest transition towards more energy (in), towards less energy followed by more energy (adjacent), or towards less energy (end) in the window of interest specified by the pipeline storage element.

Optimal Sub Dur Oracle (Presenter, Adjuster): Receives notification when a new event becomes active, and renders a sliding gradient highlight in the oracle area indicating the optimal time and last-optimal time based on the length of the subtitle string. This oracle uses the formula 0.2 sec + 0.06 sec * character, as disclosed by one of the professional subtitlers. On adjust, this oracle sets the time in the pipeline only if the current time is off by more than twice the computed standard deviation.

Video Keyframe Oracle (Preprocessor, Presenter, Adjuster): Preprocesses the loaded video by scanning for key frames. Key frames are stored in a map and are rendered as yellow lines in the oracle presentation area. On adjust, if proposed times are within a user-defined threshold distance of a key frame, the times will snap to either side of the key frame.

In addition, I intend to implement an Adjacent Splitter Oracle in a future release: such an oracle splits the previous end and next start times to

a minimum separation to prevent visual "smearing" or "direct blitting:" the minimum separation and direction of separation will be a user preference, but would typically evaluate to 2 video frames.

3.3. Choice of Platform

I would have preferred to create a platform-agnostic tool so that users on multiple platforms could use SubTek. This is the approach taken by Sabbu [1], for example. However, my system employs several different technologies that have traditionally resisted platform-independence. The user interface must include an audio waveform view and a live video preview with dynamic subtitle overlay. Although only one Video View and Script View are displayed in the beta 1 implementation,[7] future versions of SubTek may demand additional video views for multiple frames side-by-side, multiple video loops side-by-side, zoom, pan, color manipulation, or detection of mouse clicks on specific pixels.

As an overriding concern, most subtitlers use Windows machines because existing subtitling software is Windows-based, and because Windows has a mature multimedia API through DirectShow [14]. Therefore, SubTek is implemented on Microsoft Windows using the Microsoft Foundation Classes, Direct3D, DirectShow, and "i18n-aware" APIs such as those listed in National Language Support [15]. The proceeding discussion of system design

will therefore use Windows-centric terminology to distinguish between platform-independent concepts and specific implementations.

Targeting a specific platform has distinct advantages which must not be overlooked. Each platform and abstraction layer maintains its distinct object metaphors, but an abstraction layer on top of multiple platforms may implement the "lowest common denominator" of these objects. SubTek takes advantage of some Windows user interface controls, for example, for which there may be no equivalent on another platform. Since performance and accuracy are also at a premium in SubTek, coding to one platform allows for the greatest precision with the least performance hit on that platform. For example, the base unit for time measurement in SubTek is REFERENCE_TIME (TIME_FORMAT_MEDIA_TIME) from Microsoft DirectShow, which measures time as a 64-bit integer in 100ns units. This time is consistent for all DirectShow objects and calls, so no precision is lost when getting, setting, or calculating media times. Furthermore, conversions between other units, such as SMPTE drop-frame time code and 44.1kHz audio samples, can use REFERENCE_TIME as a consistent intermediary. SubTek attempts to present a consistent user experience as other applications designed for Windows, which should lead to a shallower learning

[7] More specifically, SubTek beta 1 displays only one Script frame. Multiple script views are supported in the frame via splitter windows, as evident in Figure 9.

curve for users of that platform and greater internal reliability on interface abstractions.

3.4. Data Storage, Transformation, and Consistency

As illustrated in Figure 6, the Script object is at the center of interactions between many other components, many of which are multithreaded or otherwise change state frequently.

One major design decision was to store SubTek Event objects—an event corresponds to a subtitle, command, or other notification—in C++ Standard Template Library lists rather than arrays or specialized data structures. While this decision might at first seem to invoke some serious drawbacks, it has led to several optimizations and conveniences that still permit execution in constant time while preserving the validity of iterators to unerased list members. In SubTek, most objects and routines that require Event objects also have access to an event object iterator sufficiently close to them need to discover event objects in close proximity to it.

Rather than relying on the Microsoft Foundation Classes' CView abstraction, which requires a window to operate, I implemented my own Observer design pattern [9] to ensure data consistency throughout all SubTek controls and user interface elements. The Observer is an abstract class with some hidden state, declared inside of the class being observed. Objects that wish to observe changes to an Event object, for example, inherit

from Event::Observer. When either the observer or the subject are deleted, special destructors ensure that links between the observer and the observed are verified, broken, and cleaned up safely.

SubTek supports *event transforms*, *event filters*, and *event transform filters*, mentioned briefly before and shown in Figure 5. Filters are function objects [16], or simulated closures [17], that are initialized with some state. They are used to filter events wherever subsets of Event objects from the main Script object are constructible, and they are used to manipulate, ramp, or otherwise modify event objects in response to requests from the user. For example, a time offset and ramp could be encapsulated in an event transform; SubTek could then apply this transform to a subset of events, or to the entire event list in the Script object.

3.5. Design of User Interface

The user interface is the key to the system's popularity with users: given the history of poorly-thought user interface design in the subtitling world, I tried to address the topic thoroughly while keeping the main thrust of development on the user interface that touched on the timing and oracle subsystems.

The Script View implementation is quite unusual in the subtitling world, in that it uses highly-customized rows of subclassed Windows common controls and custom-designed controls. By default, the height of each row is

three textual lines. In beta 1, the controls handle most but not all expected functionality. Customized painting and clipping routines prevent unnecessary screen updates or background erasures. Although the Script View has to manage the calculation of total height for scrolling purposes, the view can otherwise process events in amortized constant time, or in linear time in the number of visible rows, rather than linear time in the number of events in the script.

For example, because Event objects are stored in a list, it was a natural choice for the Script View to maintain records of its rows in lists as well. Each row in the list stores an iterator to the event being monitored: the iterator stores the Event's position on the Script object's event list, in addition to its ability to access the Event by reference. If the user selects a different filter for the view, it is trivial to apply the filter when iterating forwards or backwards until the next suitable iterator is found.

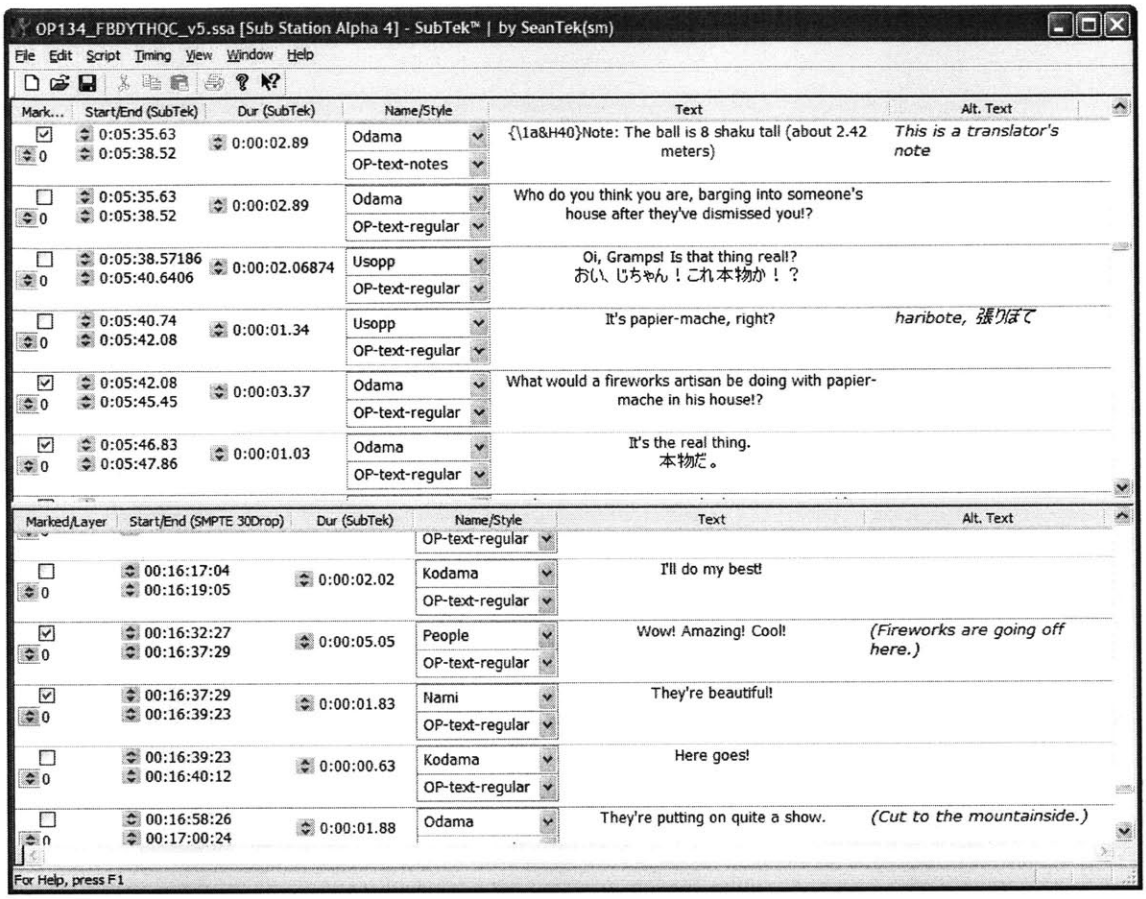


Figure 9: Script View in action. Note that the Script Frame actually supports multiple views; each row is adapts seamlessly to the event iterator that it stores.

The Video View is divided into several regions: the toolbar, the seek bar, the video, oracle display, and the status bar. Since the VMRAP9 manages the inner view (as mentioned previously), oracle and video drawing fall under the same routine. The Sub Queue oracle takes advantage of this feature, for example, by drawing the active queue items onscreen at presentation time. Figure 10 illustrates the video view with all oracles active, tying the user into a large feedback loop that culminates with the oracle adjustment phase of the On-the-Fly Timing subsystem.

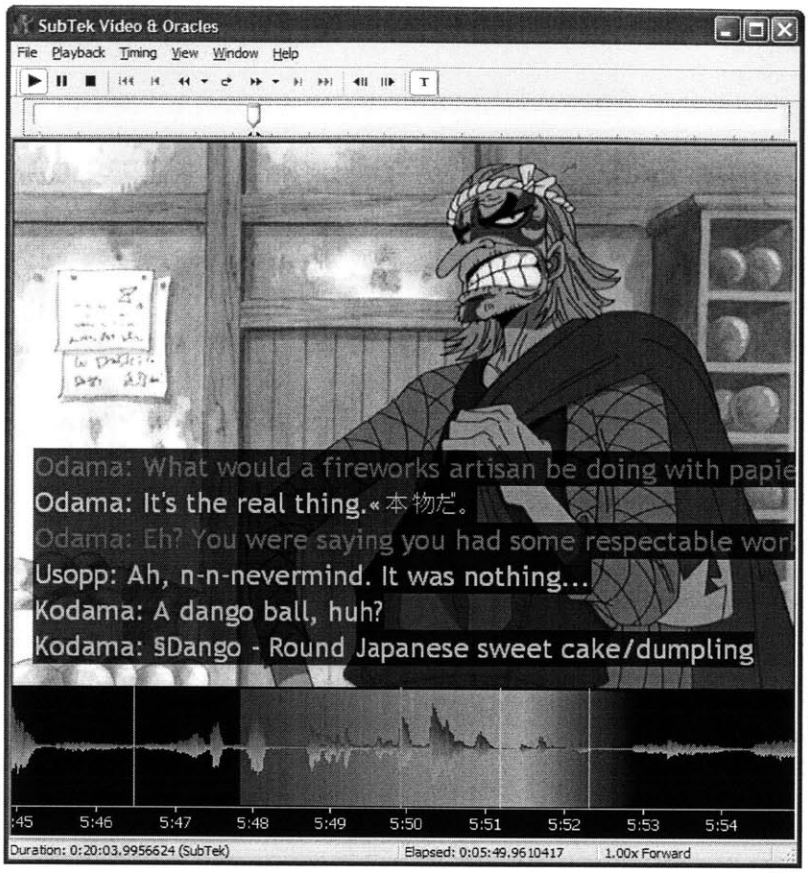


Figure 10: Video View in action. All oracles that present visual data are active: optimal sub duration, audio waveform, video key frame, and sub queue oracles.

3.6. Internationalization and Localization

SubTek is meant to be both *internationalized*—the application can work on computers around the world and process data originating from other computers around the world—and *localized*—the user interface and data formats that it presents are consistent with the local language and culture [18].

Modern Windows applications running on Windows 2000 and Windows XP can use Unicode® to store text strings. The Unicode standard assigns a unique value to every possible character in the world; it also provides

specifies encoding and transformation formats to convert between various Unicode character representations . Characters in the Basic Multilingual Plane have 16-bit code point values, from 0x0000 to 0xFFFF, and may be stored as a single unsigned short. However, higher planes code point values through 0x10FFFF, requiring the use of *surrogate pair* [20]. Where necessary, SubTek also supports these surrogate code points and the UTF-32 format, which stores Unicode values as single 32-bit integers. Internationalization features are evident, for example, in the mixed text of the Script View (Figure 9) and the Video View (Figure 10), both above.

Although some scripts are stored in binary format,[8] most scripts are stored as text with special control codes. Consequently, the encoding of the text file may vary considerably depending on the originating computer and country. SubTek relies on the Win32 API calls *MultiByteToWideChar* and *WideCharToMultiByte* to transform between Unicode and other encodings [21]. SubTek queries the system to enumerate all supported character encodings, and presents them in customized Open and Save As dialogs for script files. Since these functions rely on operating system support, they add considerable functionality to the system without the complexity of a bundled library file.

[8] The version of SubTek described herein supports limited reading of Microsoft Excel® files.

Windows executables store much of their non-executable data in resources, which are compiled and linked into the .exe file. Importantly, resources are also tagged with a locale ID identifying the language and culture to which the data corresponds; multiple resources with the same resource ID may exist in the same executable, provided that their locale IDs differ. Calls to non-locale-aware resource functions choose resources by using the caller's thread locale ID. I decided to exploit this functionality by setting SubTek's thread locale ID on application initialization; the thread locale ID is set to a user-specified value persisted to the registry. One disadvantage of this approach is that resources still have to be compiled directly into the executable; users cannot directly provide custom strings in a text file, for example. On the other hand, advanced users with access to the source code may compile localized resources in as desired, and there are technical advantages to bundling all of one's data resources in one place: it is very unlikely that resources will get lost or divided, and the application does not have to search the file system for needed files.

4. System Evaluation

System evaluation was conducted in three phases: the exploratory phase, the development and debugging phase, and the beta testing phase.

4.1. Exploration, Development, and Debugging

From the outset, SubTek has been designed with multiple audiences in mind. To that end, I tried to find at least one or two practitioners from each group: fan, professional, academic, and novice, with whom I would regularly ask questions and solicit input on desired features and levels of complexity. These users included two professional translators for the Japanese animation industry, a professor of modern Japanese culture, and a fan translator for a *fansub* group (a fan group that releases subtitled material, particularly anime, to other fans), and two video engineers involved with broadcast and surveillance applications. Novice users had a tendency to pick up the software metaphors rather quickly, so I had to find over six users in total to provide feedback throughout the first two stages of development.

Alpha testers were more than happy to provide unstructured but detailed feedback on the software development. Their comments would invariably stem from two sources: a) their needs for particular features based on other software, hardware, and other instrumentation that they use and needed to interact with, and b) their skill set conditioned on previous software use, if any.

One of the most consistent requests received, for example, was for a time edit control with more shortcut functions and data entry conveniences. The current time edit control is a textbox that allows freeform entry of times in the current time format: if the string does not match the current format,

the string is replaced with the latest one. While this was already planned, most alpha testers who espoused this concern compared the SubTek implementations to other time controls that supported per-part updating. Updating that control is on the feature list for the next beta of SubTek.

Professional translators and subtitlers maintained a fairly extensive list of features they would have liked to see, but the most oft-requested features was SMPTE drop-frame time code [22], an hh:mm:ss:ff format for time display for video running at 29.97Hz. Based on their feedback I designed several serialization and deserialization classes to specifically handle time formats, converting between REFERENCE_TIME units, SMPTE objects that store the relevant data in separate numeric fields, TimeCode objects that store data in a frame count and an enumeration for the frame rate, and strings.

Several important bugs and functionality gaps were uncovered during this alpha testing phase, including problems related to the audio waveform presentation in the Video View. Novice users were quick to point out general usability problems, such as screen flicker or partially-functional controls. These problems were addressed without great difficulty but might not have been fixed otherwise.

As SubTek began to provide useful functionality and as the subsystems were assembled and put together, I provided alpha versions to these core

users to explore. Alpha testers were instructed generally to "play with the software," but specifically, to open video files and script files, to scroll through the document, and to scrub through the video to make sure that the program did not malfunction. Most of those users who claimed that malfunctions noticed them when attempting to load a video. However, all but one system eventually parsed and opened the reference video files during alpha testing, so I deferred further investigation of these problems until after completion of the first beta.

4.2. Beta Testing Phase

Over the course of this master's thesis I contacted or was contacted by many users interested in subtitling and the development of SubTek. Of these, about 35 users formed a core group for beta testing evaluation. This group was charged with three primary tasks:

- Complete a preliminary survey indicating their interest in the software and their previous levels of experience
- Install SubTek, confirm that SubTek works with the reference files, and time the subtitles
- Complete a post-use survey indicating which parts of testing worked, which did not, and how much time was spent timing two sample media clips

Users must have completed Survey #1: Preliminary Survey in order to receive the software and supporting files. This survey consists of very general statements to get to know the user's experience, so that the user can be appropriately matched when comparing their feedback trials with other users.

Initially I planned on conducting very elaborate tests with my software. However, as the project neared beta phase, it became clear that at least some of the savings gained by reduced time timing works would be offset by the initial shock and experience gap of using the software. Therefore, videos required were reduced from three to two, and novice users were no longer required to time the full, 20-minute Japanese animation clip given the language gap for many of them. In close testing with some alpha testers, for example, I noticed that volunteers with little to no experience subtitling, and little to no experience in the video production realm, would begin to set times arbitrarily instead of stopping or submitting the second survey as-is.

Based on these circumstances, users were assigned to time *Structure and Interpretation of Computer Programs*, "Overview: An Introduction to Lisp," Lecture 1A Part 1—Text Section 1.1, of about 27 minutes [23]. Users with at least some contact with Japanese were asked to time *One Piece* Episode 134: "I'll Make it Bloom! Manly Usopp's Eight Shaku Ball," of about

20 minutes. Users with little to no Japanese exposure were asked to time approximately three minutes of *One Piece* 134, from about 00:01:50 to 00:04:40 (one minute fifty seconds to four minutes forty seconds).

Before conducting deployment evaluation, users were asked to follow a series of specific predefined steps in order to verify that the subtitling system worked on his/her system, and to familiarize themselves with the operation of SubTek. The instructions given to users are in the following section.

Post-testing user feedback required the user to complete a larger survey called Survey #2: Post-Use Survey, where they indicated exactly how much time each trial took, and any frustrations that they may have had. As part of post-testing user feedback, the user uploaded the timed subtitle files that they made.

The post-use survey asked the following information in the form of yes/no questions, where users provided their responses as checkboxes:

- Confirm that you attempted to perform, to the best of your ability, the tasks outlined in the instructions
- Did you see the views?
- Did you see the subtitle queue? Did you see the subtitles?
- If you have any video files that play successfully in Windows Media Player, did you load them into the Video View, and did they play successfully?

- Were you able to load and see the video, the video keyframes, and the audio waveform of Lecture 1A?
- Were you able to load and see the video, the video key frames, and the audio waveform of One Piece 134?

Most of the questions that followed regarded duration: the first set asked how long it took to get acclimated to the software layout, to the video controls, to the script controls, and to the controls used for on-the-fly timing, using the instructions provided. The second batch of questions asked how long it took to time the scripts using on-the-fly timing in one pass, and then as many subsequent passes or manual edits as necessary to get the script to an "acceptable" state. In all cases, users were provided small sections of a timed script in order to see what the results were supposed to look like, but were instructed not to copy from it. A timer was provided on the HTML survey form, although users could use any timer they wished. Users were further asked in the survey how accurate they thought their reported times were for their first-pass and final versions.

I qualitatively compared random samples of their subtitled work against the reference scripts and the unsubtitled video.

4.3. User Instructions

The following instructions were distributed to users in text-based form, along with appropriate URLs, pictures, and much more detailed steps:

In this phase, you will install SubTek and supporting materials, will confirm or attempt to confirm that the reference files open and play back successfully in SubTek, and will time two untimed scripts.

First, you should make sure that you have or are in the process of downloading the following files or packages:

- The SubTek beta prerelease from Sean Leonard
- The reference files: one English lecture, one Japanese animation, three scripts for the English lecture, and at least two scripts for the Japanese animation
- If you wish to see timed subtitles displayed onscreen during video playback, you need to install a preview filter. One popular preview filter is DirectVobSub [24]. To install it, go to the VobSub website and download VSFilter 2.36. The archive should have a file called VSFilter.dll. Extract this file to your hard drive and register it with regsvr32.

Extract the SubTek beta files to a folder of your choice.

If you own a ShuttleXpress or Shuttle PRO jog/shuttle control, you may use the included preferences file to configure your Shuttle product with SubTek. The shuttle performs fast and slow forward and reverse. The jog advances or goes back by frame. The third button toggles between play and pause, and the second button performs "instant replay."

Start SubTek by opening SubTek.exe.

Two windows will display: the script view and the video view.

4.3.1. Tutorial: Script View and Video View

The script view is where you load, save, and edit scripts directly. This script will be used when you are performing timing in the video view. Go to File->Open to open a script. You will see a figure like the left dialog box of Figure 11.

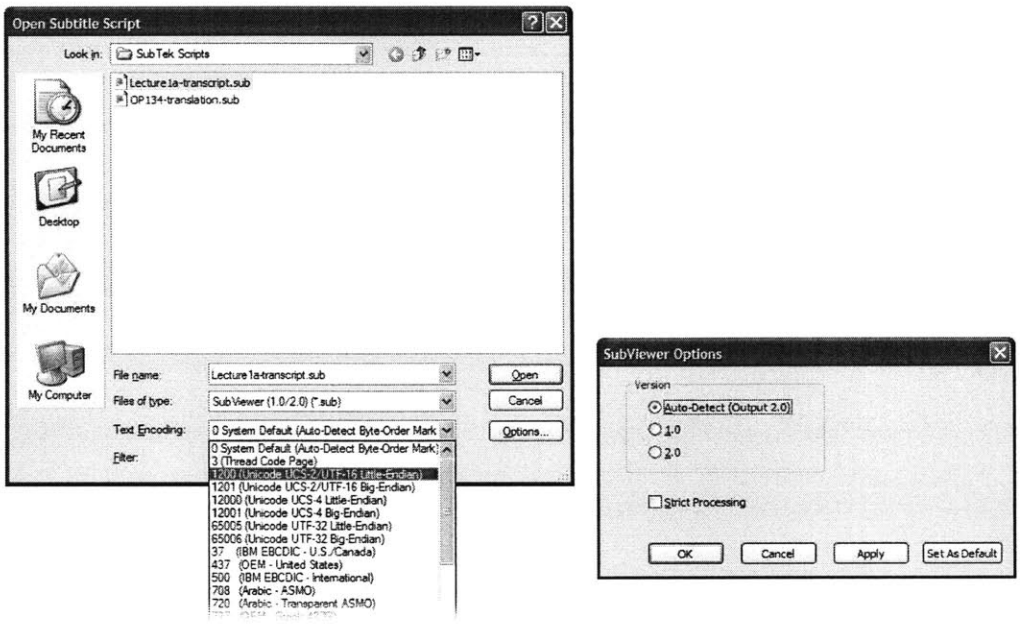


Figure 11: Open Script Dialog, showing text encoding drop-down list and specific format options dialog

Choose the acclimation script for the English lecture. You should notice that the script view now contains many rows.

Examine the script view so that you have a reasonable sense of what each field means. Try to drag the headers or track the dividers (clicking and dragging on the space between the headers). If a field does not seem very

useful to you, you can compress the header to the left and make that column disappear. However, keep at least the Time, Dur, and Text columns visible for the following exercises. Scroll upwards and downwards: the data follows the scrolling action.

Change a text item. Press Tab. Change an alternate text item.

In the Time field, the first time is the start time and the time beneath that is the end time. Click into one of the end times and increase the number by one second or one-hundredth of a second. You may need to use backspace. If you entered a valid time, you will see the time for "Dur" change to reflect the new duration of the subtitle. You can manipulate the start and duration fields for similar effects.

Click on one of the "Name" fields: it is the combo box with "Hal" in it. You can choose a name from the drop-down list, or you can type in a different name. Since "Hal" is the only speaker in this lecture, however, the drop-down list only displays "Hal."

Enter your name, and move to another name field above or below with the mouse. Check the drop-down list. You should find your name listed as one of the possibilities.

If you press the "up" or "down" arrows while in the combo box, you will automatically select the previous or next name in the list. Try it.

Now, try to save the file in a different format by going to File->Save As. Under file type, choose "SubViewer." Notice that the Options button on the right became enabled: that is because the SubViewer format can be customized a bit.

Click "Options." The SubViewer dialog box pops up, as illustrated in Figure 11 above. If you want to change the specific format options, you can in this dialog box. Click "OK" or "Cancel," depending on what you want to do.

Now type a different file name in the text box, and click "Save." The title bar of Script View should now indicate that SubViewer is the current file format when saving to disk (it is in brackets). If you choose to Save instead of Save As, the file will be written in SubViewer format.

Switch to the video view, but keep the script view open. You can drag the script view window out of the way a little if necessary. In video view, go to the File menu, then click "Open Video/Audio 1...." Open the Lecture-1a video file and confirm that the file loads and starts playing successfully. This may take some time.

While the file is playing, try navigating through the video. You should be able to use the seek bar to go to the middle of the video. Pressing "Pause" will pause the video; pressing the fast forward button should increase the rate of the video to a certain point. Notice that at the bottom, you can see the duration, the currently-elapsed time, and the rate of playback.

While the video is playing, go back to the script view. Find the first time in the script, which should be around 28 minutes. Once you confirm the exact time, go back to the video view and use the seek bar to move to that position in the video clip. Hal should be talking about "learning Lisp now." Watch the video for a few seconds, and also notice that there is status information at the bottom of the screen. This information should look like an audio waveform and yellow lines corresponding to the video key frames. The magenta line is the current position. When a yellow line crosses the magenta line, should almost always see a shot or fade in the video above.

Now, press the Instant Replay button, which is between fast reverse and fast forward. By default, the video goes back about five seconds, but keeps on playing if you are in playback mode. Now, try pressing the spacebar: it also triggers Instant Replay.

Try increasing the rate of playback by pressing fast forward once or twice. You can also click on the down arrow next to fast forward and pick a rate. Press P. The video pauses. Press P again. The video continues playing back, but at 1x rate.

Press Y. The video pauses. Press Y again; nothing happens. Y is for pause only.

If you need to change the rate quickly, you can press the upper row of buttons, from ~, through the number keys, through backspace. Those keys

correspond to the specific menu items in the Forward and Reverse drop-down menus.

Now look under the view menu. There are two distinct options: Load Preview Filter, and Realtime Script Preview. By default, the first is checked. If VobSub or another provider is installed, you can click on Display Realtime Preview to see the subtitles in the script view dynamically laid over the video view. Try it now. Note, however, that you will only see subtitles when the subtitles are scheduled to display onscreen. If you are sure that you should be seeing a subtitle at that particular time in the video, try to make sure that VobSub or another provider is loaded.

By default, VobSub "prebuffers" subtitles for display, which saves many CPU cycles overall but may also cause VobSub to skip certain subtitles because they are not ready yet. You can turn off prebuffering if you are experiencing these problems: check the VobSub documentation for details.

If all of these instructions and the results made sense to you, you understand the basics of how to use SubTek's script and video views.

4.3.2. Tutorial: On-the-Fly Timing

Let's try some on-the-fly timing. First, turn off Realtime Script Preview. This will make the system more responsive, and in any case, you want to create timed subtitles, not watch pre-timed subtitles.

Move the script window so that it is onscreen and so that you can see the subtitles and times at the top. Switch to the video window, though, leaving the script window visible. Rewind the video back to the beginning of the second lecture, before Hal starts talking about "learning Lisp now."

Press the T button on the toolbar: you are now in on-the-fly timing mode. You should see the current event from the top of the list in the script view, except the text of that event is now onscreen on the video view. Other events are also below it, in Name: Text format. The text should be surrounded by a partially transparent, dark box. The "current event" should be in green. If you do not see something similar to this, then something is wrong.

Now, let's actually set some times. It does not matter what is going on screen right now—just press and hold "J" for several seconds. Notice that the event turned red, just like in Figure 10. You set a time when you pressed "J," and now the event is "hot."

After several seconds, release "J" and watch the screen. You will notice that the event is no longer "hot," and that the item beneath it is now in green. That next item is now "active."

Try pressing "J" again, then release. Notice how the event advances.

By pressing and releasing "J," you are setting times for those events.

While the video is still playing and timing mode is still on, make sure that you can see the script view. See if you can find those events that you just timed. If you look closely at the times, you may notice that they have changed.

With the script view visible, look back at the video view and see if you can identify the next event. It is green. Now, look back at the script view and see if you can find that same event. Once you have found the event, go back to the video view while you can still see the event in the script view. The video should be playing.

Press "J" for awhile, then release. You should see the times in the script view change almost instantly.

If all of these instructions and the results made sense to you, you understand the very basics of timing. "J" sets and releases times.

Now, let's try to time some events. At this point, you have already timed events, but the times are most likely incorrect. Rewind the video to the beginning of the second lecture.

You will notice that the list of events on top of the video does not change. To move them backwards, press "L." Notice that the active event goes back one. You can use your fourth finger while your index finger is hovering over "J." Keep on pressing "L" until you get back to the first event.

Now the video should be playing and Hal should start talking momentarily. You can tell if he is going to talk because of the audio waveform underneath the video. As close to the moment when he begins speaking the active line, press "J." Release "J" when he finishes speaking that subtitle line.

If you are ready to continue, then keep on watching Hal and press "J" when he speaks the line; release "J" when he stops speaking.

If you get behind, you can catch up without setting times. Press ";" with your pinky (or any finger). The active event moves forward, but no time is set. Keep on pressing and releasing that until you catch up.

You may have noticed that Hal sometimes stops talking but then starts speaking the next subtitle line almost immediately. For those intervals, you can press "K" while you hold "J."

Try it now. After pressing "K," you can release either "K" or "J"—neither will have any effect. Look closely at the event list. The preceding event now has a "[Q]" in it. That means that the previous event was added to a queue of partially-timed events. When you release the last button (either "J" or "K"), all of the events in the queue, including the currently "hot" one, will be analyzed and their times will be set.

Alternating between "J" and "K" can be really useful for rapid dialogues or long monologues. Try it with Hal's lecture for awhile so that you understand how it works, even if he pauses for some time between speaking.

You can constantly hold "J" down and keep on pressing "K," you can constantly hold "K" down while pressing "J," or you can alternate between "J" and "K." However, once "J" and "K" are both released, the queue of events is processed.

Do you remember that pressing the spacebar performs Instant Replay? This is really useful if you make a mistake. Try pressing "J" (or "K," since they do the same thing) and releasing it. After releasing it, pretend that you made a mistake that you want to correct.

Press spacebar; you just went back five seconds. If the subtitle that you missed was particularly long, you may need to press spacebar multiple times.

Since the active event has been set to the next event, you will need to go back by pressing "L." So, press "L" and wait for the right moment to set the event time again.

(Advanced hint: in that situation, you can also press "H." "H" functions like pressing "L," then immediately pressing "J." That might be too much to remember now, though, so just stick with "J" and "K" until you master them. There is also an "N" key, but you can figure out "N" later.)

If all of these instructions and the results made sense to you, you understand just about everything required for on-the-fly timing in SubTek.

Try going back and seriously timing a minute of Hal's second lecture. After you are done, turn off timing mode, and turn on Realtime Script Preview.

Rewind and check your times. Are the times accurate, or do they seem to be a little off? If they are not perfectly accurate, do not worry: perfection comes with practice.

4.3.3. Task 1: Time Lecture 1A Part I

Once you feel comfortable with the controls for on-the-fly timing, open the timed script excerpt of Hal's first lecture. Watch it closely with realtime preview on so that you get an idea of the rhythm.

Once you feel prepared, open the untimed script of Hal's first lecture.

Turn off Realtime Script Preview (it will not really help anyway), and turn on timing mode.

Begin timing the script, and begin timing yourself. You should try to go through in one pass, trying to avoid Instant Replay where possible. However, use Instant Replay whenever a quick replay could save a lot of editing time later.

If you find that your technique seems terribly off, stop early and restart from the beginning. You can reset your time, but only if you stop before you begin memorizing parts of the lecture.

Once you reach the end of the first lecture, stop timing mode. Record how long it took to go through the first pass.

Save the document with a different name, particularly a name that includes "-afterfirstpass" or something. The file format should be Sub Station Alpha v4.

Now, you need to get the times to match Hal's spoken dialogue precisely. Hopefully the oracle and on-the-fly timing subsystems will have taken care of everything, so you may simply wish to play your work back and return to script view to make periodic changes. You may use any combination of techniques that you wish to in SubTek, but do not spend more than an hour-and-a-half adjusting the times. If you reach an hour-and-a-half and are not finished, record how far along you were able to get in the script, and enter this data in the survey.

Save your latest script in Sub Station Alpha v4 format under a different file name, such as "-final." Submit this file with your survey.

4.3.4. Task 2a: Time all of One Piece 134 in Japanese

Once you feel comfortable with the controls for on-the-fly timing, open the timed script excerpt of One Piece 134. Watch it closely with realtime preview on so that you get an idea of the rhythm.

Once you feel prepared, open the untimed script of One Piece 134.

Turn off Realtime Script Preview (it will not really help anyway), and turn on timing mode.

Begin timing the script, and begin timing yourself. You should try to go through in one pass, trying to avoid Instant Replay where possible. However, use Instant Replay whenever a quick replay could save a lot of editing time later.

If you find that your technique seems terribly off, stop early and restart from the beginning. You can reset your time, but only if you stop before you begin memorizing parts of the lecture.

Once you reach the end of One Piece 134, stop timing mode. Record how long it took to go through the first pass.

Save the document with a different name, particularly a name that includes "-afterfirstpass" or something. The file format should be Sub Station Alpha v4.

Now, you need to get the times to match the spoken dialogue precisely. Hopefully the oracle and on-the-fly timing subsystems will have taken care of everything, so you may simply wish to play your work back and return to script view to make periodic changes. You may use any combination of techniques that you wish to in SubTek, but do not spend more than an hour-and-a-half adjusting the times. If you reach an hour-and-a-half and are not

finished, record how far along you were able to get in the script, and enter this data in the survey.

Save your latest script in Sub Station Alpha v4 format under a different file name, such as "-final." Submit this file with your survey.

4.3.5. Task 2b: Time three minutes of One Piece 134 in Japanese

Once you feel comfortable with the controls for on-the-fly timing, open the full timed script of One Piece 134. Watch it closely with realtime preview on so that you get an idea of the rhythm and the storyline. Search online for *One Piece* to learn more about the main characters and memorize what they look like. Pay particular attention to the sequence between 0:01:50 and 0:04:40. You may watch it twice, but try not to completely memorize the script. (It is unlikely that you can, in any case, if you understand no Japanese.)

Once you feel prepared, open the untimed script excerpt of One Piece 134 between 0:01:50 and 0:04:40. This script is simplified from the primary script: the subtitles are individually longer in length and duration, so you will not have to guess at word or clause breaks in Japanese as much.

Turn off Realtime Script Preview (it will not really help anyway), and turn on timing mode.

Begin timing the script, and begin timing yourself. You should try to go through in one pass, but you may use Instant Replay somewhat more

liberally if you become totally lost in the Japanese. If you merely missed the cue, though, you should use Instant Replay sparingly as with the Hal lecture.

Once you reach the end of One Piece 134, stop timing mode. Record how long it took to go through the first pass.

Save the document with a different name, particularly a name that includes "-afterfirstpass" or something. The file format should be Sub Station Alpha v4.

Now, you need to get the times to match the spoken dialogue precisely. Hopefully the oracle and on-the-fly timing subsystems will have taken care of everything, so you may simply wish to play your work back and return to script view to make periodic changes. You may use any combination of techniques that you wish to in SubTek, but do not spend more than an half-an-hour adjusting the times. If you reach an half-an-hour and are not finished, record how far along you were able to get in the script, and enter this data in the survey.

Save your latest script under a different file name, such as "-final." The file format should be Sub Station Alpha v4. Submit this file with your survey.

4.4. Feedback and Future Directions

Although all 35 users responded to initial calls, not all were able to respond in time with fully-completed survey results. At minimum, SubTek requires a 2GHz Pentium-4 class computer or higher if running the MPEG-4

decoder, two mp3 audio decoders (to extract presentation and audio waveform data), three avi splitters (to extract presentation, video key frame, and audio waveform data), and all oracles. If the Preview Filter connects to DirectVobSub and all run in parallel, a 2.6GHz to 3GHz machine is necessary for smooth playback. On a 1.7GHz laptop, reported one user, the software would run but would periodically desynchronize between audio and video, severely affecting the quality of his timing. SubTek also heavily depends on stable libraries being installed: some users failed to load the video file due to irresolvable codec incompatibilities, and thus could only comment on the script processing features of SubTek.

Nevertheless, 20 users were able to respond with structured feedback. Of these 20, users claimed an average of 10 minutes for acclimation to the software, and 1.9 hours to time the English lecture. Unsurprisingly, novice users reported longer times: one as long as an hour for the full acclimation process. Alpha testers, who had seen the software in its previous incarnations, took very little time acclimating to the software.

Of the 1.9 hours, most users spent less than half of the time (0.75 hours, or 45 minutes) on the first pass through the system. No users performed the first pass in exactly real time, although one professional managed to complete the first pass in less than 34 minutes. The later pass

required more time for everyone: the nearly universal complaint was the non-fully-implemented time control.

Not all users completed the One Piece 134 task: only 5 users completed the 3-minute version, and 12 attempted and completed the full 20-minute episode. Four of the five users finished the first pass in less than four minutes; on average, it took the five users about 13 minutes to complete the task. The remaining twelve users averaged 1.5 hours, instead of the reported 3.7 hours for a similar task using other tools.

From a qualitative standpoint, users expressed satisfaction with SubTek's performance overall, and were glad to see the different approach that SubTek takes to gathering times. Fansubbers and professional translators expressed their desire to use the software on new animation series that they will be working on, so it is encouraging to see that SubTek is already establishing a permanent user base. I have resolved to keep in touch with these users and to continue to develop SubTek. In light of my desire to encourage development of a permanent user base, the confidentiality and license agreements (Appendix C) were modified so that the current beta testers may use it as long as they desire until it is finally released under a public, open-source-style license.

Despite these promising results, I remain skeptical that these ratios are sustainable across a wide variety of media, and more controlled trials

should be performed to obtain more consistent results. The English lecture was essentially a long monologue: the lack of background noise and music clearly helped the audio processing filter; however, it also meant that natural lengthy pauses were few and far between. Finally, the window of accuracy for acceptable subtitle start and end times depend on where lines are broken and where subtitles are split: this typesetting factor becomes much more prominent in monologues, because the decision to break between lengthy pauses versus grammatically-appropriate boundaries is ultimately resolved by the transcriptionist's aesthetic sense. Since I transcribed the audio, I consciously gravitated towards more lengthy pauses when splitting long sentences across subtitles. It is not unreasonable to assume that a slightly different presentation style or subtitle format could have adversely affected the results.

As a fansub production, One Piece 134's script came pre-segmented and preformatted; I made only ancillary changes to aggregate extremely short or merely partially-translated subtitles. The nature of the programming material differed completely from the English lecture on computer science, and so in that sense it provided good contrast. One Piece's script—and scripts for fansubs more generally—tend to include very tight time cuts as part of the fansub aesthetic. In contrast, the information density provided through a non-fictional classroom recording naturally corresponds with larger and more

lingering subtitles. Given additional time, I hope to find other kinds of content to see if there is a medium between these extremes.

Before working on the feature set for beta 2, I intend to examine the performance issues and bottlenecks, and to improve the audio processing module by performing some preprocessing of the audio data before the user streams the files for on-the-fly timing. Outstanding user interface concerns, such as the time control and inaccessibility of preferences, will also be addressed. Format conflicts and codec incompatibilities also plague end-user software, so creating a "complete package" of required library files and supporting executables would probably aid testers tremendously.

I conclude that the SubTek experiment was a success, and that it serves as a good starting point and framework for future work.

5. Analysis and Conclusion

This work has several implications for the different audiences that it can possibly reach. The performance of the software ultimately exceeded my expectations, as I initially estimated that SubTek would improve the process by no more than an hour or an hour and a half, at maximum. Although 1.5 to 2 hours is still a substantial time commitment per 22-minute episode, these figures—including the figures from novice users—suggest that timing need not be a laborious and mundane process. The process is livened when users actively participate in a subtitle's making.

As the goal of SubTek is to assist the human timer and not replace him or her, it is far too early to pronounce any changes in the fabric of fandom, academia, or industry on account of this software. Fansubbers routinely embellish their subtitles with puns, cultural explanations, and translations of written Japanese in the video; my system does not attempt to replicate these features, nor does my system, in its current incarnation, automate the process of visual placement for signage and text effects. While this tool may help academics subtitle obscure works with ease, too few pure scholars participated in this round of beta testing to draw meaningful conclusions.

Activity already seen in these four spheres suggests that distribution of these transcripted, translated and timed works is already changing in an era of increasing globalization. As seasoned translators or sophisticated translation processes begin to work with SubTek, I hope that their translation and subtitling activities will further the course of exciting cultural growth [25].

6. References

[1] Kryptolus, "Sabbu," [online document], 2005, [cited 2005 Sep 11], Available HTTP: http://www.sabbu.com/en/index.html

[2] Medusa, "Medusa," [online document], 2005, [cited 2005 Sep 11], Available HTTP: http://sourceforge.net/projects/medusa

[3] S. Shults, "XombieSub," [online document], 2005, [cited 2005 Sep 11], Available HTTP: http://xombie.soldats.net/index.htm

[4] J. Morones, "Lemony video subtitler," [online document], 2005, [cited 2005 Sep 11], Available HTTP: http://www.jorgemorones.com/lemony/

[5] SoftNI, "Subtitler," SoftNI, [Online document], 2005, [cited 2005 Sep 11], Available HTTP: http://www.softni.com/subtitler.html

[6] X. Huang, F. Alleva, H.-W. Hon, M.-Y. Hwang, K.-F. Lee, R. Rosenfeld, "The Sphinx-II speech recognition system: an overview (http://sourceforge.net/projects/cmusphinx/)" *in Computer Speech and Language*, [online document], 1992 Jan. 15, [cited 2004 May 7], Available HTTP: http://citeseer.ist.psu.edu/huang92sphinxii.html

[7] T. Ezzat, G. Geiger, and T. Poggio, "Trainable Videorealistic Speech Animation," *in Proceedings of SIGGRAPH 2002*, [online document], 2002, [cited 2004 May 7], Available HTTP: http://www.ai.mit.edu/projects/cbcl/publications/ps/siggraph02.pdf

[8] J. Wang and T.-S. Chua, "A Framework for Video Scene Boundary Detection," *in Proceedings of SIGGRAPH 2002*, [online document], 2002, [cited 2004 May 7], Available HTTP: http://portal.acm.org/citation.cfm?id=641055&dl=ACM&coll=GUIDE

[9] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[10] T. Ottinger, "What Makes a Good Object-Oriented Design?" *ootips*, [online document], 1998, [cited 2005 Sep 11], Available HTTP: http://ootips.org/ood-principles.html

[11] R. Martin, "The Interface Segregation Principle," *C++ Report*, [online document], 1996, [cited 2005 Sep 11], Available HTTP: http://www.objectmentor.com/resources/articles/isp.pdf

[12] R. Martin, "The Dependency Inversion Principle," *C++ Report*, [online document], 1996, [cited 2005 Sep 11], Available HTTP: http://www.objectmentor.com/resources/articles/dip.pdf

[13] A. V. Oppenheim and A. S. Willsky, with S. H. Nawab, *Signals and Systems*, Prentice-Hall, Second Edition, 1997, p. 205.

[14] Microsoft, "DirectShow," *MSDN Library*, [online document], 2005, [cited 2005 Sep 11], Available HTTP: http://msdn.microsoft.com/archive/en-us/directx9_c/directx/htm/directshow.asp

[15] Microsoft, "National Language Support," *MSDN Library*, [online document], 2005, [cited 2005 Sep 11], Available HTTP: http://msdn.microsoft.com/library/en-us/intl/nls_19f8.asp

[16] Wikipedia, "Function object," *Wikipedia*, [online document], 2005, [cited 2005 Sep 11], Available HTTP: http://en.wikipedia.org/wiki/Function_object

[17] Wikipedia, "Closure (computer science)," *Wikipedia*, [online document], 2005, [cited 2005 Sep 11], Available HTTP: http://en.wikipedia.org/wiki/Closure_(computer_science)

[18] Wikipedia, "Internationalization and localization," *Wikipedia*, [online document], 2005, [cited 2005 Sep 11], Available HTTP: http://en.wikipedia.org/wiki/Internationalization_and_localization

[19] The Unicode Consortium, "What is Unicode?" *Unicode*, [online document], 2005, [cited 2005 Sep 11], Available HTTP: http://www.unicode.org/standard/WhatIsUnicode.html

[20] The Unicode Consortium, "Special Areas and Format Characters," in *The Unicode Standard, Version 4.0*, Addison-Wesley, Boston, MA, 2003, pp. 383-409.

[21] Microsoft, "Unicode and Character Set Functions," *MSDN Library*, [online document], 2005, [cited 2005 Sep 11], Available HTTP: http://msdn.microsoft.com/library/en-us/intl/unicode_19mb.asp

[22] Wikipedia, "SMPTE time code," *Wikipedia*, 2005, [cited 2005 Sep 11], Available HTTP: http://en.wikipedia.org/wiki/SMPTE_time_code

[23] H. Abelson and G. J. Sussman, "Structure and Interpretation of Computer Programs, Video Lectures," 1986, [cited 2005 Sep 11], Available HTTP: http://swiss.csail.mit.edu/classes/6.001/abelson-sussman-lectueres/

[24] Gabest, "VSFilter 2.36," *guliverkli*, 2005, [cited 2005 Sep 11], Available HTTP: http://sourceforge.net/projects/guliverkli/

[25] S. Leonard, "Progress Against the Law: Fan Distribution, Copyright, and the Explosive Growth of Japanese Animation," 1.10, 2004 Apr. 29, [cited 2004 May 10], Available HTTP: http://web.mit.edu/seantek/www/papers/

Appendix A: Survey #1

The first survey referenced in System Evaluation (p. 43) is provided in this appendix. The surveys were originally written as web pages, and results were submitted to me by cgiemail.

SubTek™ - The System for Rapid Subtitling | by SeanTek℠

Thank you for beta testing SubTek. Beta testing consists of four distinct steps:

1. Contacting Sean Leonard at **seantek@mit.edu** and agreeing to the confidentiality and software license terms for SubTek. This step should take approximately fifteen minutes.
2. Completing Survey #1, "Preliminary Survey," below. This survey asks general questions about your previous subtitling experience. Beta testers with any level of experience are welcome. This step should take approximately ten minutes.
3. Receiving the software and supporting instructions, and testing the software on your system by following the instructions to create timed subtitle files. Depending on your specific instructions, this step may take anywhere from half an hour to four hours. You should allot sufficient time to read through the instructions and subsequent survey, so that you know what to expect.
4. Completing Survey #2, "Post-Use Survey." This survey asks specific questions about your use of SubTek and allows beta testers to provide detailed evaluations of the software. As part of the survey process, you will submit the timed scripts that you created in step 3, for further analysis.

Structured User Feedback Survey #1: Preliminary Survey

Before receiving and testing SubTek, you need to provide some information about your subtitling experience so that your results can be evaluated more objectively. All information will be held in strict confidence. You may, at your option, wish to submit this survey over a secure connection (HTTPS/SSL). Connecting to this website via https provides a slightly higher level of security than submitting these responses over the web in plaintext; howsever, e-mail communications will not, by default, be encrypted.

Items marked with * are required.

Identifying Information:

*** Name:** Alyssa P. Hacker

Alias (?):

*** E-mail (?):**

Other contact info (?): East Campus

* ☑ Please check this box to confirm that you have already received the confidentiality and software license terms for SubTek, that you have already agreed to them,and that you have already transmitted your acceptance to Sean Leonard. This surveywill not be accepted until you can check this box.

Demographics and Experience:

* Check all boxes that apply to describe your degree of involvement with subtitling (you must check at least one):

- ☐ I am just a volunteer for this project--I have little to no involvement with subtitling (this option is exlusive with most other options)
- ☐ I am a "subtitle buff" (I prefer subtitled films, television, animation, etc.)
- ☑ I am a gamer
- ☑ I have another, unlisted affiliation that draws me to subtitling: I like to hack
- ☐ I am a fansubber
  - ☐ I am a digisubber
- ☐ I am a translator
  Which languages from/to (one per line)?
- ☐ I am a professional in a media industry
  - ☐ I am a professional in the anime or Asian culture industries
  - ☐ I am a professional in the video games industry
- ☑ I am a student
  - ☐ I am a high school student
  - ☑ I am a college student
  - ☐ I am a graduate student
- ☐ I am a post-doctoral fellow
- ☐ I am a professor
- ☐ I am a researcher with an interest in subtitles for specific goals (*e.g.*, subtitles of German Wartime Cinema)
- ☐ I am a reseracher with an interest in subtitling (*i.e.*, the technical or linguistic problems of subtitling)

My academic discipline is:

Languages spoken and degree of proficiency:

*** English:**

- ○ I speak no English
- ○ I have minimal contact with English
- ○ I have some contact with English
- ○ I have some training in English
- ○ I have considerable training in English (*e.g.*, a year or two in college)
- ○ I am proficient in English
- ○ I am fluent in English (non-native speaker)
- ◉ I am fluent in English (native speaker)
- ○ I am highly fluent in English and am comfortable with complex linguistic tasks such as translation **to** English (**from** another language) and editing of English with a high degree of linguistic accuracy and cultural fidelity

*** Japanese:**

- ○ I speak no Japanese
- ○ I have minimal contact with Japanese
- ◉ I have some contact with Japanese
- ○ I have some training in Japanese
- ○ I have considerable training in Japanese (*e.g.*, a year or two in college)
- ○ I am proficient in Japanese
- ○ I am fluent in Japanese (non-native speaker)
- ○ I am fluent in Japanese (native speaker)
- ○ I am highly fluent in Japanese and am comfortable with complex linguistic tasks such as translation **from** Japanese (**to** another language, but particularly English)

and editing of Japanese with high degree of linguistic accuracy and cultual fidelity

Subtitling Software Experience

*** Previous Experience:**

◉ I have no substantial experience with subtitling software.
If you have no experience with existing subtitling software, you are done. Please submit this form below.

○ I have enough experience that I can answer the questions below.

Which subtitling tools do you use on a regular basis?

Please write one software item per line. If the version number is important, then please note the version number; otherwise, the name alone is sufficient. Try to write the programs in the order in which you use them, from concept to completion:

Some popular software (for example):
VobSub, VirtualDub, Sub Station Alpha, Microsoft Excel, Notepad, SoftNI, Avisynth, JWPce

Please report the estimate of how long it takes, using the software listed above, for you to perform the following tasks on a single television-episode-length (~22 minute) media clip in your *own* language (transcription). If you do not know, do not routinely perform these tasks, or do not perform these tasks in independent stages, leave these text boxes blank. Please enter your response in minutes:

To transcribe the text:

To time the transcribed text:

To transcribe and time the text:

To perform file conversions:

To render the subtitles to final, non-textual form:

To complete the whole subtitling process, from start to finish:

How accurate are the results that you just provided?

Choose Accuracy

Please report the estimate of how long it takes, using the software listed above, for you to perform the following tasks on a single television-episode-length (~22 minute) media clip in *another* language (translation). If you do not know, do not routinely perform these tasks, or do not perform these tasks in independent stages, leave these text boxes blank. Please enter your response in minutes:

To translate the text:

To time the translated text:

To transcribe and time the text:

To perform file conversions:

To render the subtitles to final, non-textual form:

To complete the whole subtitling process, from start to finish:

How accurate are the results that you just provided?

Choose Accuracy

What are your primary frustrations with these existing tools that you use or have used? (Check all that apply, or "Other" and fill in)

- ☐ Takes/took too long to learn
- ☐ Unstable software
- ☐ Undocumented software
- ☐ Unsupported software
- ☐ Takes too long to time
- ☐ Takes too long to translate
- ☐ Unsupported file formats
- ☐ Unsupported time formats
- ☐ Unsupported languages
- ☐ Poor user interface
- ☐ Not enough features (karaoke, text effects) that I need
- ☐ Too many features (karaoke, text effects) that I do not need; not enough features that I want
- ☐ Other

Submit

If you would like to be cc'd with these survey responses, please enter your address here (note that the e-mail will not be encrypted, regardless of how you accessed this site):

Submit Reset

Preliminary Survey: Before Using SubTek

Post-Use Survey: SubTek Usage Report and Feedback

Return to SubTek Home

Last updated 8/29/2005 by Sean Leonard

For comments and suggestions, send e-mail to **seantek@mit.edu**

Return to Tektropolis Home

Appendix B: Code

Since the code base for this thesis comprises nearly 33,000 gross source lines of code (31,129 net), it is impractical to reprint the code in its entirety here. Instead, the interfaces and definitions for the On-the-Fly Timing subsystem and Oracle subsystem are presented, because these two subsystems form SubTek's most important contributions over previous work in the field. The declaration files are heavily commented so that their interfaces are clear, well-defined, and unambiguous.

IOracle.h

```
// IOracle.h: Interface declarations for oracles: these interfaces
// define the characteristics of an oracle in
// the System for Rapid Subtitling.
// by Sean Leonard
// Declaration file.
// last modified: 8/7/2005

#ifndef __SUBTEK_IORACLE_H_
#define __SUBTEK_IORACLE_H_

#if defined(_MSC_VER) && (_MSC_VER >= 1020)
#pragma once
#endif

#include <d3dx9core.h>
#include <deque>
#include <list>
#include "STEvent.h"

/**
 * It is possible for an encapsulating application to customize
 * and redefine the pipeline storage elements by declaring
 * the following token. The interface would lose binary compatibility,
 * but would retain its organizational structure.
 */
#ifndef __SUBTEK_IORACLE_CUSTOM_PIPELINE_STORAGE_
// standard pipeline storage

#include <vector>
#include <map>
#include <functional>

struct PIPELINE_STORAGE_TIMES {
   REFERENCE_TIME rtEndPrev;     // the PREVIOUS ENDING time
   REFERENCE_TIME rtStart;       // the CURRENT start time.
   // default constructor
   PIPELINE_STORAGE_TIMES(REFERENCE_TIME rtEP = -1LL,
      REFERENCE_TIME rtS = -1LL) : rtEndPrev(rtEP), rtStart(rtS)
   {}
```

```
};
struct PIPELINE_STORAGE_ELEMENT {
   PIPELINE_STORAGE_TIMES times; // primary times.
   REFERENCE_TIME rtStdDev;
   // window: rtMin < rtMax always (unless both invalid)
   REFERENCE_TIME rtMin;
   REFERENCE_TIME rtMax;
   // alternate times, sorted in DESCENDING order. It is assumed that a
   // time pair as equally valid as the main times pair will have
   // confidence level 0.0;
   // times with less confidence should have confidence levels < 0.0.
   std::map<double, PIPELINE_STORAGE_TIMES,
      std::greater<double> > altTimes;
   // default constructor
   PIPELINE_STORAGE_ELEMENT(REFERENCE_TIME rtEP = -1LL,
      REFERENCE_TIME rtS = -1LL, REFERENCE_TIME rtSD = -1LL,
      REFERENCE_TIME rtMn = -1LL, REFERENCE_TIME rtMx = -1LL)
      : times(rtEP, rtS), rtStdDev(rtSD), rtMin(rtMn), rtMax(rtMx)
   {}
};

struct PIPELINE_STORAGE_PROPERTIES;
struct TIMING_PROPERTIES {          // properties of the timing
subsystem
   std::list<STEvent>::iterator iter_end;
   std::reverse_iterator<std::list<STEvent>::iterator> iter_rend;
   bool (*pFilterFunc)(const STEvent &ste);
};

struct TIMING_SIGNAL_PROPERTIES {
   const std::deque<std::pair<std::list<STEvent>::iterator,
      REFERENCE_TIME> > *pQue;
   std::list<STEvent>::iterator *pActiveEvent;
   int nAdvance;
};

#else
// forward definitions
   #ifndef PIPELINE_STORAGE_PROPERTIES
      struct PIPELINE_STORAGE_PROPERTIES;
   #endif
   #ifndef PIPELINE_STORAGE_ELEMENT
      struct PIPELINE_STORAGE_ELEMENT;
   #endif
   #ifndef TIMING_PROPERTIES
      struct TIMING_PROPERTIES;
   #endif
   #ifndef TIMING_SIGNAL_PROPERTIES
      struct TIMING_SIGNAL_PROPERTIES;
   #endif
#endif

// {5524F89B-A62C-408C-94AD-325164A26A6F}
/*DEFINE_GUID(IID_IOraclePreprocessor,
```

```
0x5524f89b, 0xa62c, 0x408c,
0x94, 0xad, 0x32, 0x51, 0x64, 0xa2, 0x6a, 0x6f); */
/*
// {5524F89B-A62C-408c-94AD-325164A26A6F}
static const GUID <<name>> =
{ 0x5524f89b, 0xa62c, 0x408c,
{ 0x94, 0xad, 0x32, 0x51, 0x64, 0xa2, 0x6a, 0x6f } };
*/

// {A7EF76A1-B24B-4f52-A3D0-035F1EBCAE02}
/* DEFINE_GUID(IID_IOraclePresenter,
0xa7ef76a1, 0xb24b, 0x4f52,
0xa3, 0xd0, 0x3, 0x5f, 0x1e, 0xbc, 0xae, 0x2);

// {A7EF76A1-B24B-4f52-A3D0-035F1EBCAE02}
static const GUID <<name>> =
{ 0xa7ef76a1, 0xb24b, 0x4f52,
{ 0xa3, 0xd0, 0x3, 0x5f, 0x1e, 0xbc, 0xae, 0x2 } };
*/

// {44F3FD5A-B794-4fb9-8019-F130CDC3DF85}
/*DEFINE_GUID(IID_IOracleAdjuster,
0x44f3fd5a, 0xb794, 0x4fb9,
0x80, 0x19, 0xf1, 0x30, 0xcd, 0xc3, 0xdf, 0x85); */
/*
// {44F3FD5A-B794-4fb9-8019-F130CDC3DF85}
static const GUID <<name>> =
{ 0x44f3fd5a, 0xb794, 0x4fb9,
{ 0x80, 0x19, 0xf1, 0x30, 0xcd, 0xc3, 0xdf, 0x85 } };
*/
/*static const GUID IID_IOracleAdjuster =
{ 0x44f3fd5a, 0xb794, 0x4fb9,
{ 0x80, 0x19, 0xf1, 0x30, 0xcd, 0xc3, 0xdf, 0x85 } }; */


// This interface preprocesses
[uuid("5524F89B-A62C-408C-94AD-325164A26A6F")]
interface IOraclePreprocessor : public IUnknown
{
public:
   // when a media clip is first loaded, the source filter (which
   // should but may not actually have the IFileSourceFilter interface)
   // is passed so that the oracle can preprocess the file. This
   // function blocks until preprocessing is complete; however, the
   // oracle may actually run preprocessing on a separate thread.
   // It is expected that the source filter will be in a
   // fully-connected graph. While preprocess may do anything to the
   // source filter and the graph, it is expected that Preprocess will
   // not actually run the graph: if it needs to extract some
   // or lots of data, it may create a "shadow graph" and run through
   // the shadow graph, extracting values as necessary
   // (on a separate thread).
   // INPUT: an interface array containing the current sources being
   // processed. When a source is unloaded or otherwise closed, it is
```

```
    // expected that the application will call Preprocess again and
    // specify NULL for that source. The length of CInterfaceArray
    // should not change; however, the max length ever received will
    // be considered the max number of "slots" for preprocessing
    // sources; nonpresent values (because of short count) will be
    // considered NULL. When one of the spots is NULL, it is expected
    // that the preprocessor will unload or otherwise
    // stop preprocessing that source. Specifying NULL for the array
    // pointer results in all sources being unloaded.
    // It may be desirable for the application to specify some NULL
    // sources that are actually not NULL, if the application does not
    // want the preprocessor to process them.
    virtual HRESULT STDMETHODCALLTYPE Preprocess(
        CInterfaceArray<IBaseFilter> *pSources) PURE;
};


// IOraclePresenter
[uuid("A7EF76A1-B24B-4F52-A3D0-035F1EBCAE02")]
interface IOraclePresenter : public IUnknown
{
public:
    // each oracle presenter needs:
    // >>the ID3DXRenderToSurface (from which it can infer the
    // description/format and the device)
    // >>the VMR9PresentationInfo, from which it can infer the
    // start and stop times. It can also examine the surface,
    // the presentation flags, and the aspect ratio if it wishes.
    // Attempting to change these values is possible but not
    // necessarily recommended; in any case, PresentImage is done
    // with them, so any changes would only mess with their
    // interpretation by subsequent oracles on this pass only.
    // >>the hint to the boundary between video and oracle
    // presentation (i.e., the D3DRECT specifying the rectangle
    // for presentation, which normally only differs from the
    // device->desc by a change of the top pixel from 0 to the
    // start of the oracle region). However, the oracle presenter is
    // NOT required to respect this boundary: it may wish, for
    // example, to draw directly on top of the video to present
    // additional data, suggest special effects,
    // or highlight certain regions.
    // The oracle presenter should draw:
    // at transformed z-val MaxZ (1.0) to present in the
    // foreground (the default)
    // at transformed z-val MinZ (0.0) to present in the background
    // somewhere in between to show-off fancy 3D effects.
    virtual HRESULT STDMETHODCALLTYPE Present(
        IDirect3DSwapChain9 *pSwapChain,
        VMR9PresentationInfo *lpPresInfo,
        D3DRECT *pRect, REFERENCE_TIME rtStart, REFERENCE_TIME rtEnd,
        REFERENCE_TIME viewStart, REFERENCE_TIME viewEnd,
        FLOAT coordStart, FLOAT coordEnd, D3DMATRIX *pMatFocus) PURE;
};
```

```
// IOracleAdjuster
[uuid("44F3FD5A-B794-4fb9-8019-F130CDC3DF85")]
interface IOracleAdjuster : public IPersist
{
public:
   // Called by the application/on-the-fly timing subsystem
   // to inform the oracle that timing has started.
   // Most oracles will not modify the list of active events;
   // however, it is possible that some oracles may wish to display
   // events other than the currently active event.
   // The list must be present in order to conduct timing, and
   // the list must be the same list that the active and hot events
   // are on. However, the list may be empty.
   // Hence, we only need the ends of the list.
   virtual HRESULT STDMETHODCALLTYPE NotifyStartTiming(
      const TIMING_PROPERTIES *pTimingProps) PURE;

   // Called by the subsystem to inform
   // the oracle that a new event is now HOT (the event at the back
   // of the deque is the hot one). Note that no matter what, the
   // queue type is fixed: the queue always is a deque
   // that holds STEvents and reference times.
   // also provides the active event, which is USUALLY the same as
   // the hot event, but not always (for example, when timing first
   // commences, and when a user requests some processing of another
   // sequence of events while looking at a totally different event)
   // if there is no active event, pActiveEvent may be NULL. If there
   // are no queued or hot events, pQue may be NULL or may point to
   // an empty deque.
   virtual HRESULT STDMETHODCALLTYPE NotifySignalTiming(
      const TIMING_SIGNAL_PROPERTIES *pTSProps) PURE;

   /**
    * Called by the on-the-fly timing subsystem to inform
    * an oracle that timing adjustments will commence or will terminate.
    * Parameter: a pointer to the properties of the pipeline storage.
    * If this parameter is NULL, the oracle will release or free any
    * references it holds to the pipeline storage.
    */
   virtual HRESULT STDMETHODCALLTYPE AdvisePipelineStorage(
      const PIPELINE_STORAGE_PROPERTIES *pProps) PURE;

   /**
    * Called by the on-the-fly timing subsystem repeatedly
    * to make adjustments using the oracle's capabilities.
    * The oracle examines the pipeline storage with respect to the
    * current event and stage, and returns adjusted start or end times.
    * See the PIPELINE_STORAGE_ELEMENT structure (a part of the
    * interface standard) for details.
    * prtAdjEnd and prtAdjStart initially reflect the ideal time from
    * the perspective of the on-the-fly timing subsystem. This is
    * frequently--but not always--the result of the previous stage. The
    * timing subsystem may modify the result of the previous stage (for
    * example, choosing one of the alternate times) and pass that value
```

```
 * in prtAdjEnd/prtAdjStart if the subsystem has some specialized
 * knowledge.
 * In most scenarios, the first-stage call to Adjust will have the
 * prtAdjEnd/prtAdjStart times taken directly from the event deque.
 * For initial-start times (iEvent == 0), only prtAdjStart is valid.
 * For final-end times (iEvent == deque.size()), only prtAdjEnd
 * is valid.
 * For adjacent times (all other times), both prtAdjEnd and
 * prtAdjStart are valid and should be written-to. However, prtAdjEnd
 * and prtAdjStart may or may not be the same time.
 * Parameters:
 * [in] iEvent: the index of the event. This value takes on the range
 * [0, event queue size], INCLUSIVE, so that the end point has the
 * event index queue size.
 * [in, out] pElement: a pointer to memory of a pipeline storage
 * element, but not necessarily an element in the storage array. This
 * storage element initially contains the values that need adjusting.
 * The output (depending on the result code) contains the values that
 * have been adjusted.
 * Return:
 * HRESULT codes that the pipeline controller must support:
 * S_OK: the oracle adjusted the times and wrote the adjusted times
 * and supporting statistics to pElement.
 * S_FALSE: the oracle examined the times and determined that no
 * adjusting was needed. The element at pElement was NOT modified,
 * but the orcale examined the data nevertheless (hence, the element
 * should be considered "processed").
 * (other success codes): whatever happened, the data in pElement
 * is valid.
 * E_POINTER: The pointer to pElement was invalid.
 * E_INVALIDARG: At least one of the indexes iEvent or iStage
 * were invalid. The element at pElement was not modified.
 * E_NOTIMPL: This oracle does not adjust times. The element at
 * pElement was NOT modified.
 * E_PENDING: The data necessary to complete the operation is
 * not yet available. (This may happen, for example, because
 * of buffer starvation while processing the media clip.)
 * The element at pElement is in an undefined state.
 * This error code suggests that given enough elapsed time,
 * the data may become available for the oracle
 * to process a future request.
 * VFW_E_NOT_RUNNING: The oracle never got a chance to preprocess
 * or filter the data, so it knows nothing of the source
 * material. However, this error may occur after the oracle
 * conducts some preliminary calculations, so the element
 * at pElement is in an undefined state.
 * E_FAIL: An unspecified failure has occured. The element at
 * pElement is in an undefined state.
 * (other error codes) Whatever happened, the element at pElement
 * is in an undefined state.
 */
virtual HRESULT STDMETHODCALLTYPE Adjust(
   ULONG iEvent, ULONG iStage,
   PIPELINE_STORAGE_ELEMENT *pElement) PURE;
```

```
    // Called by the on-the-fly timing subsystem
    // to inform the oracle that timing has started.
    virtual HRESULT STDMETHODCALLTYPE NotifyStopTiming(
        /* include some state of the timing system */)
    PURE;

    // Oracle Adjuster information-getter functions.
    // GetClassID is already required by IPersist. But note that
    // it is valid for a class to return E_FAIL and not deliver.
    // Also, if the oracle implements a filter via
    // IBaseFilter, it has to provide IPersist functionality anyway.

    // TODO: not sure if this is necessary or not. CONSIDER a function
    // like QueryAdjusterInfo.
    // In theory, queries the oracle for ADJUSTER info. This is different
    // from general oracle information. For example, it may specify
    // (as flags) that it adjusts based on the audio stream, adjusts
    // based on the video stream, adjusts based on pre-existing
    // subtitles, or adjusts via user-supplied parameters or files.
    // (However, the oracle may PRESENT data inconsistent with these
    // flags... in which case we'd want QueryPresenterInfo in
    // the other interface.)
    // virtual HRESULT STDMETHODCALLTYPE QueryAdjusterInfo(
    //     LPDWORD pInfo);
};

#ifndef __SUBTEK_IORACLE_CUSTOM_PIPELINE_STORAGE_

struct PIPELINE_STORAGE_PROPERTIES
{
    // double-vector that stores all pipeline values. The outer
    // (first) vector indexes rows while the inner (second) vector
    // indexes columns; hence, every iteration through the deque
    // of events will traverse an inner vector.
    std::vector<std::vector<PIPELINE_STORAGE_ELEMENT> > pipelineStorage;
    // need the list of oracles in the pipeline
    CInterfaceArray<IOracleAdjuster> adjusters;

    // default constructors (C++)
    PIPELINE_STORAGE_PROPERTIES() {}
    // points are the number of points (events + 1, or time "in and out"
    // points) to process stages are the number of adjusters to have, and
    // thus the number of stages to process per point.
    PIPELINE_STORAGE_PROPERTIES(size_t nPoints, size_t nStages) :
     pipelineStorage(nPoints)
    {
        adjusters.SetCount(nStages);
        for (size_t p = 0; p < pipelineStorage.size(); p++) {
            pipelineStorage[p].resize(nStages);
        }
    } // end utility constructor
};
```

```
#endif

#endif
```

Oracles.h : CVideoKeyframeOracle

```
class CVideoKeyframeOracleFilter;

[uuid("893C9F64-A9E9-49c7-866F-08D99FCF50FF")]
class CVideoKeyframeOracle : public CUnknown, CAMThread,
   public IOraclePresenter, IOracleAdjuster, IOraclePreprocessor
{
   friend class CVideoKeyframeOracleFilter;
public:
   // constructor
   CVideoKeyframeOracle(LPUNKNOWN pUnk, HRESULT *phr);
   virtual ~CVideoKeyframeOracle();

   // completing the implementation of CUnknown
   DECLARE_IUNKNOWN;
   STDMETHODIMP NonDelegatingQueryInterface(REFIID riid, void **ppv);

   // implementation of IOraclePreprocessor
   STDMETHODIMP Preprocess(CInterfaceArray<IBaseFilter> *pSources);

   // implementation of IOraclePresenter
   STDMETHODIMP Present(IDirect3DSwapChain9 *pSwapChain,
VMR9PresentationInfo *lpPresInfo,
      D3DRECT *pRect, REFERENCE_TIME rtStart, REFERENCE_TIME rtEnd,
      REFERENCE_TIME viewStart, REFERENCE_TIME viewEnd,
      FLOAT coordStart, FLOAT coordEnd, D3DMATRIX *pMatFocus);

   // implementation of IOracleAdjuster
  STDMETHODIMP NotifyStartTiming(const TIMING_PROPERTIES *pTimingProps);
   STDMETHODIMP NotifySignalTiming(const TIMING_SIGNAL_PROPERTIES
*pTSProps);
   STDMETHODIMP AdvisePipelineStorage(const PIPELINE_STORAGE_PROPERTIES
*pProps);
   STDMETHODIMP Adjust(
      ULONG iEvent, ULONG iStage,
      PIPELINE_STORAGE_ELEMENT *pElement);
   STDMETHODIMP NotifyStopTiming();

   STDMETHODIMP GetClassID(CLSID *pClassID) {
      if (!pClassID) return E_POINTER;
      *pClassID = __uuidof(this);
      return S_OK;
   }

   void Exit();   // turns off the thread.

protected:
   // preferences
```

```
   D3DCOLOR m_col;            // color of the lines for
"keyframes"/"syncpoints"
   // data on all of the keyframes
   std::map<REFERENCE_TIME, REFERENCE_TIME> m_keyframes;
   std::map<REFERENCE_TIME, REFERENCE_TIME>::iterator m_lastinsert; //
an optimization
   CCritSec m_KFLock;            // a lock on the keyframe map.
Coordinates reading and writing.
   REFERENCE_TIME m_rtMin;       // minimum valid time. Defaults to -1
(no times valid)
                            // before this point, this oracle cannot
provide reasonable
                            // information about the existence of
keyframes.
   REFERENCE_TIME m_rtMax;       // maximum valid time. Defaults to -1
(no times valid)
                            // at and after this point, this oracle
cannot provide
                            // reasonable information about the existence
of keyframes.
   const PIPELINE_STORAGE_PROPERTIES *m_pProps; // you can't modify it.

   // Thread issues
   DWORD ThreadProc();
   CComPtr<IBaseFilter> m_pOriginalSourceFilter;   // refreshed every
time the thread runs, providing the original source filter.
                            // this value is guaranteed to be valid until
CallWorker returns the first time.
   IBaseFilter *m_pOracleFilter; // used internally as the renderer/sink
that generates keyframe results.
   bool m_bSafeToWaitAndClose;      // safe to wait and close?
};
```

Oracles.cpp : CVideoKeyframeOracle

```
// Video Keyframe Oracle

[uuid("602774D3-4B13-4973-8428-D2CFC3590FE4")]
class CVideoKeyframeOracleFilter : public CBaseRenderer
{
public:
   CVideoKeyframeOracleFilter(LPUNKNOWN pUnk, CVideoKeyframeOracle
&rOracle, HRESULT *phr);
   virtual ~CVideoKeyframeOracleFilter() {}

   // implementation of pure virtual functions in CBaseRenderer filter
   HRESULT CheckMediaType(const CMediaType *pmt);
   // HRESULT Transform(IMediaSample *pSample);
   HRESULT DoRenderSample(IMediaSample *pMediaSample);
protected:
   CVideoKeyframeOracle &m_rOracle;
};
```

```
CVideoKeyframeOracleFilter::CVideoKeyframeOracleFilter(LPUNKNOWN pUnk,
CVideoKeyframeOracle &rOracle,
                                                HRESULT *phr) :
CBaseRenderer(__uuidof(this), NAME("CVideoKeyframeOracleFilter"),
              pUnk, phr),
m_rOracle(rOracle)
{
   // set the name (this is optional)
/* ASSERT(m_pName == NULL);
   WCHAR name[] = L"Video Keyframe Oracle Sink";
   DWORD nameLen = lstrlenW(name)+1;
   m_pName = new WCHAR[nameLen];
   lstrcpynW(m_pName, name, sizeof(name)/sizeof(WCHAR)); */
}

HRESULT CVideoKeyframeOracleFilter::DoRenderSample(IMediaSample *pSample)
{
   ASSERT(pSample);
   // note that we COULD try GetMediaTime, but most filters
   // don't stamp 'em that way.
   std::pair<REFERENCE_TIME, REFERENCE_TIME> rt;
   HRESULT hr = pSample->GetTime(&rt.first, &rt.second);
   ASSERT(SUCCEEDED(hr) || hr == VFW_E_SAMPLE_TIME_NOT_SET); //
VFW_E_SAMPLE_TIME_NOT_SET 0x80040249L
   if (hr == VFW_E_SAMPLE_TIME_NOT_SET) {
      // that's not good.
      return S_FALSE;
   }
  // technically, the above are STREAM times. Now make them media times.
   rt.first += m_pInputPin->CurrentStartTime();
   rt.second += m_pInputPin->CurrentStartTime();
   // okay; now they are start and end times.
   // pSample->IsDiscontinuity();
   // pSample->IsPreroll();
   hr = pSample->IsSyncPoint();
   if (hr == S_OK) {
      typedef std::map<REFERENCE_TIME,
            REFERENCE_TIME>::iterator rtiter;
      // okay. Add to the map.
      hr = pSample->IsDiscontinuity();
      bool added = false;
      { CAutoLock l(&m_rOracle.m_KFLock); // alrightie folks, time to
lock down
      if (hr == S_OK) {
         // because ths sample is a discontinuity, our hint of
m_lastinsert is invalid.
         std::pair<rtiter, bool>
inserted(m_rOracle.m_keyframes.insert(rt));
         if (inserted.second) {
            m_rOracle.m_lastinsert = inserted.first;
         }
         added = inserted.second;
      } else {
         ASSERT(hr == S_FALSE);
```

```
            // since it's not a discontinuity, we can ATTEMPT to insert
            // in amortized constant time
            rtiter prev(m_rOracle.m_lastinsert);
           m_rOracle.m_lastinsert = m_rOracle.m_keyframes.insert(prev, rt);
            added = (prev != m_rOracle.m_lastinsert);
         }}
         if (added) {
/*          TRACE(_T("Keyframe %.4g->%.4g added. Yay!\n"),
               (double)rt.first / (double)STUtil::s,
               (double)rt.second / (double)STUtil::s
               ); */
         } else {
/*          TRACE(_T("Keyframe %.4g->%.4g not added: conflict!\n"),
(double)rt.first / (double)STUtil::s,
               (double)rt.second / (double)STUtil::s); */
         }
      } else {
         ASSERT(hr == S_FALSE);
      }
      return S_OK;
}

HRESULT CVideoKeyframeOracleFilter::CheckMediaType(const CMediaType *pmt)
{
   GUID g = *(pmt->Type());

   if (g == MEDIATYPE_Video || g == MEDIATYPE_Timecode || g ==
MEDIATYPE_AnalogVideo
      || g == MEDIATYPE_Interleaved)
   {
      if (pmt->bTemporalCompression) {
         // HARDCODED: yellow color.
         m_rOracle.m_col = 0xd7fcfb13;
         return S_OK;
      } else {
         // TODO: okay for now, but we probably will get all frames
         // as syncpoints/keyframes, and that's just not informative
         // in this case, let's set the syncpoints/"keyframes" to a very
faint gray color.
         // HARDCODED: faint gray color.
         m_rOracle.m_col = 0x4f909090;
         return S_OK;
      }
   }

   return VFW_E_TYPE_NOT_ACCEPTED;
} // end CheckInputType



// Video Keyframe Oracle Constructor
CVideoKeyframeOracle::CVideoKeyframeOracle(LPUNKNOWN pUnk, HRESULT *phr)
: m_rtMax(-1LL), m_pProps(NULL), CUnknown(NAME("Video Keyframe Oracle"),
pUnk, phr),
```

```
m_pOracleFilter(NULL), m_bSafeToWaitAndClose(false),
m_col(0x0)
// m_segRate(1.0), m_segStart(0LL), m_segStop(0LL),
{
   if (phr && FAILED(*phr)) {
      // abort
   } else {
   using namespace std;
#ifdef max
#define maxMath max;
#undef max
   m_rtMin = numeric_limits<REFERENCE_TIME>::max();
#define max maxMath
#undef maxMath
#endif
   m_lastinsert = m_keyframes.end();

   // create the filter internally
   m_pOracleFilter = new CVideoKeyframeOracleFilter(NULL, *this, phr);
   m_pOracleFilter->AddRef();
   }
} // end Constructor

CVideoKeyframeOracle::~CVideoKeyframeOracle()
{
   Exit();
   CAutoLock l(&m_KFLock); // necessary to not "pull the rug" out from
other threads
   ULONG howmanyleft;
   if (m_pOracleFilter) {
      howmanyleft = m_pOracleFilter->Release();
      ASSERT(howmanyleft == 0);  // this is true because the thread has
closed down.
   }
} // end Destructor

void CVideoKeyframeOracle::Exit()
{
   CAutoLock ll(&m_AccessLock);
   // now, the CallWorker is not processing requests.
   // TODO: fix this. on rare instances, the application deadlocks on
close.
   if (ThreadExists()) {      // "thread exists" means "the thread was
actually started at some point"
   if (m_bSafeToWaitAndClose) {
      Close();// just in the final throes of closing (or already closed)
   } else {
      // well, since no one else is calling CallWorker, we might as well
do it now
      CallWorker(0);
      // there is one distinct corner case, which is dealt with in
ThreadProc.
      ASSERT(m_bSafeToWaitAndClose);
      Close();
```

```
    }
    }
}  // end turn off of thread

// CUnknown
STDMETHODIMP CVideoKeyframeOracle::NonDelegatingQueryInterface(REFIID
riid, void **ppv)
{
    CheckPointer(ppv,E_POINTER);
    if (riid == __uuidof(IOraclePresenter)) {
        return GetInterface((IOraclePresenter *)this, ppv);
    } else if (riid == __uuidof(IOracleAdjuster)) {
        return GetInterface((IOracleAdjuster *)this, ppv);
    } else if (riid == __uuidof(IOraclePreprocessor)) {
        return GetInterface((IOraclePreprocessor *)this, ppv);
    } else {
        return CUnknown::NonDelegatingQueryInterface(riid, ppv);
    }
} // end NonDelegatingQueryInterface

// CTransInPlaceFilter

// IOraclePresenter
STDMETHODIMP CVideoKeyframeOracle::Present(IDirect3DSwapChain9
*pSwapChain, VMR9PresentationInfo *lpPresInfo,
        D3DRECT *pRect, REFERENCE_TIME rtStart, REFERENCE_TIME rtEnd,
        REFERENCE_TIME viewStart, REFERENCE_TIME viewEnd,
        FLOAT coordStart, FLOAT coordEnd, D3DMATRIX *pMatFocus)
{
    // IDEA: render to surface.
    const REFERENCE_TIME viewDur = viewEnd - viewStart;
    const double viewDurd = (double)viewDur;
    CComPtr<IDirect3DDevice9> pDevice;
    HRESULT hr;
    hr = pSwapChain->GetDevice(&pDevice);
    CComPtr<ID3DXLine> pLine;
    hr = D3DXCreateLine(pDevice, &pLine);
    if (FAILED(hr)) return hr;

    pLine->SetAntialias(FALSE);
    pLine->SetWidth(1.0f);
    D3DXVECTOR2 oneline[2] = {
        D3DXVECTOR2(0.0f, (FLOAT)pRect->y1),
        D3DXVECTOR2(0.0f, (FLOAT)pRect->y2),
    };

    if (S_OK == (hr = pLine->Begin())) {
    CAutoLock l(&m_KFLock);
    for (
    std::map<REFERENCE_TIME, REFERENCE_TIME>::iterator
iter(m_keyframes.lower_bound(viewStart));
    iter != m_keyframes.end() && viewEnd > iter->first;
    iter++)
    {
```

```
      // okay, we're good to go.

      FLOAT coordStart = (FLOAT)((double)(iter->first - viewStart) /
viewDurd);
      FLOAT coordEnd = (FLOAT)((double)(iter->second - viewEnd) /
viewDurd);
      FLOAT xStart = (FLOAT)pRect->x1 + coordStart * (FLOAT)(pRect->x2 -
pRect->x1);
      oneline[0].x = oneline[1].x = xStart;
      // draw
      pLine->Draw(&oneline[0], 2, m_col);
   } // end iteration
   pLine->End();
   } else { // end Begin/End
      return hr;
   }
   return E_NOTIMPL;
}

// IOracleAdjuster

STDMETHODIMP CVideoKeyframeOracle::NotifyStartTiming(const
TIMING_PROPERTIES *pTimingProps)
{
   return S_OK;
}

STDMETHODIMP CVideoKeyframeOracle::NotifySignalTiming(const
TIMING_SIGNAL_PROPERTIES *pTSProps)
{
   return S_OK;
}

STDMETHODIMP CVideoKeyframeOracle::AdvisePipelineStorage(const
PIPELINE_STORAGE_PROPERTIES *pProps)
{
   CAutoLock l(&m_KFLock);
   m_pProps = pProps;
   return S_OK;
}

STDMETHODIMP CVideoKeyframeOracle::NotifyStopTiming()
{
   return E_NOTIMPL;
}

// IOraclePreprocessor
STDMETHODIMP
CVideoKeyframeOracle::Preprocess(CInterfaceArray<IBaseFilter> *pSources)
{
   // start the preprocessing thread.
   bool donotquit = false;
   {CAutoLock l(&m_KFLock);
   // for now, just take the first source that is non-NULL.
```

```
   if (!pSources || pSources->IsEmpty()) {
      Exit();
      m_keyframes.clear();
      return S_OK;
   }
   for (size_t i = 0; i < pSources->GetCount(); i++) {
      if (!pSources->GetAt(i)) {
         continue;   // it's just null.
      }
      if (m_pOriginalSourceFilter == pSources->GetAt(i)) {
         return S_FALSE;   // nothing changed.
      }
      // okay, here we go.
      m_pOriginalSourceFilter = pSources->GetAt(i);
      donotquit = true;
      m_keyframes.clear();
   }} // end the lock.
   if (!donotquit) {
      Exit();
      m_keyframes.clear();
      return S_OK;      // everything removed.
   }
   CAutoLock l2(&m_KFLock);
#ifdef max
#define max_temp max
#undef max
#endif
#ifdef min
#define min_temp min
#undef min
#endif
   m_rtMin = std::numeric_limits<REFERENCE_TIME>::max();
  m_rtMax = -999999999LL; // std::numeric_limits<REFERENCE_TIME>::min();
#ifdef max_temp
/* #define max max_temp
#undef max_temp */
#endif
/*#ifdef min_temp
#define min min_temp
#undef min_temp
#endif */
   if (Create()) {
      return CallWorker(0x01);         // this should return S_OK if
process launched; otherwise, a failure code
                                    // from the underlying function that
failed
   // WaitForSingleObject(m_ev, INFINITE);   // wait until we at least
know that
                                       // preprocessing has started
   } else {
      return E_PENDING;    // already working on something else (so we
can't work on your request; sorry).
   }
} // end Preprocess
```

```
DWORD CVideoKeyframeOracle::ThreadProc()
{
   CComPtr<IFilterGraph2> pInternalGraph;
   // DWORD dwRegister = 0x0;
   try {
      HRESULT hr;
      STUtil::HResult Hr(hr);
      Hr = pInternalGraph.CoCreateInstance(CLSID_FilterGraph, NULL);
      // ASSERT(SUCCEEDED(STUtil::AddToRot((IUnknown *)pInternalGraph,
&dwRegister)));

      // for now, we only support files.
      CComPtr<IFileSourceFilter> pFileF;
     Hr = m_pOriginalSourceFilter->QueryInterface(IID_IFileSourceFilter,
(void**)&pFileF);
      LPOLESTR pszFileName;
      AM_MEDIA_TYPE mt;
      Hr = pFileF->GetCurFile(&pszFileName, &mt);

      pFileF.Release();

      // now, seed the filter graph.
      ASSERT(m_pOracleFilter);
      Hr = pInternalGraph->AddFilter(m_pOracleFilter, L"Video Keyframe
Oracle Renderer");
      ASSERT(hr == S_OK);
      CStringW strFileName(pszFileName);
      CComPtr<IBaseFilter> pInternalFileF;
      CoTaskMemFree(pszFileName);
      FreeMediaType(mt);

      Hr = pInternalGraph->AddSourceFilter(strFileName, strFileName,
&pInternalFileF);

      // okay; done with original source filter.
      m_pOriginalSourceFilter = NULL;

      CComPtr<IEnumPins> pEnum;
      pInternalFileF->EnumPins(&pEnum);
      CComPtr<IPin> pFirstPin;
      Hr = pEnum->Next(1, &pFirstPin, NULL);
      pEnum.Release();
      // this may not be "perfect."
      Hr = pInternalGraph->RenderEx(pFirstPin,
AM_RENDEREX_RENDERTOEXISTINGRENDERERS, NULL);

   } catch (HRESULT hr) {
      ASSERT(FAILED(hr));
      DWORD first = GetRequest();
      if (first == 0x0) {
         Reply(0x0);
         VERIFY(CheckRequest(&first)); // can't use ASSERT because we
want CheckRequest to appear in release build
```

```
            Reply(hr);
        } else {
            Reply(hr);
        }
        return hr;
    } // end try catch on preparing the internal graph.

    // now we can reply
    {
        DWORD req = GetRequest();
        if (req == 0x0) {
            Reply(0x0);
            VERIFY(CheckRequest(&req));
            ASSERT(req == 0x1);
            return E_ABORT;   // aborted because we JUST shut down.
        } else {
            ASSERT(req == 0x1);
            Reply(S_OK);
        }
    }
    CComPtr<IMediaControl> pMC;
    CComPtr<IMediaEvent> pME;
    VERIFY(SUCCEEDED(pInternalGraph->QueryInterface(&pMC)));
    VERIFY(SUCCEEDED(pInternalGraph->QueryInterface(&pME)));
    HRESULT hr;
    // need to make the graph run as quickly as possible
    {CComPtr<IMediaFilter> pGraphMF;
    pInternalGraph->QueryInterface(&pGraphMF);
    hr = pGraphMF->SetSyncSource(NULL); ASSERT(SUCCEEDED(hr)); }

    hr = pMC->Run();
    long evcode;
    bool mustreply = false;
    if (SUCCEEDED(hr)) {
        // wait for completion
        DWORD requestparam = 0xFFFF;
        do {
            hr = pME->WaitForCompletion(50, &evcode);
            if (hr == VFW_E_WRONG_STATE) {
                OAFilterState fs; // how did we get here?
                hr = pMC->GetState(1, &fs);
                OutputDebugString(L"PoopWRongState\n");
                break;
            }
            if (CheckRequest(&requestparam)) {
                // the only request that we accept is STOP--that's request 0
                ASSERT(requestparam == 0x0);
                mustreply = true;
                break;
            }
        } while (hr == E_ABORT);   // abort means timeout (so we can cycle
and check any requests)
    }
    // okay; ready to close.
```

```
   m_bSafeToWaitAndClose = true;
   // okay; done with self.
   // STUtil::RemoveFromRot(dwRegister);
   hr = pMC->Stop();
   hr = pInternalGraph->RemoveFilter(m_pOracleFilter);

   pInternalGraph.Release();
   // okay: we're about to exit.
   // in a corner case, m_bSafeToWaitAndClose was false, so the
destructor tried
   // to CallWorker. So, we are processing zero requests, one request,
or two
   // requests. (Two requests: someone else called CallWorker, and the
destructor also called CallWorker.)
   // so we need to reply to as many requests as necessary.
   DWORD reqp;
   while (CheckRequest(&reqp)) {
      Reply(0);   // okay, we're stopping.
   }

   return S_OK;
}
```

SubTekTiming.cpp : On-the-Fly-Timing Subsystem (part):

```
// initialize the timing engine
// return value: true = "successfully transitioned from non-timing to
timing state"
bool CSubTekApp::StartTiming()
{
   // first of all, if no media is loaded, then we can't start timing.
   // Or if we're already in timing mode, we shouldn't try to start
again.
   if (m_bTiming || !(m_pVideoFrame && m_pVideoFrame->IsMediaLoaded()))
{
      return false;
   }
   // the media controller and seeker must be present.
   if (!m_pVideoFrame->m_pMC || !m_pVideoFrame->m_pMS) {
      return false;
   }
   if (!(m_pScriptFrame &&
      m_pScriptFrame->GetActiveDocument()-
>IsKindOf(RUNTIME_CLASS(CSubTekScript)))) {
      return false;
   }

   // choose a predicate or filter for events
   m_pFilterFunc = IsTrue;

   // load the first active event
```

```
   CSubTekScript *pScript = (CSubTekScript *)m_pScriptFrame-
>GetActiveDocument();

   // if m_activeevent is at the end, then the user may not have
explicitly put the
   // initial selection somewhere. So we PROBABLY should put it back at
the beginning.
   if (pScript->m_activeevent == pScript->m_events.end()
&& !m_bCreateEventAfterEnd) {
      pScript->m_activeevent = pScript->m_events.begin();
   }

   // initialize the active line (you can see all of the previous events
at the top, initially)
   m_iActiveLine = m_cPrev;
   // make an array for ourselves
   m_adjusters.RemoveAll();
   m_adjusters.SetCount(m_oracles.GetCount());
   size_t actualcount = 0;
   HRESULT hr = S_OK;
   TIMING_SIGNAL_PROPERTIES tsp = { NULL, &pScript->m_activeevent, 0 };
   for (POSITION pos = m_oracles.GetHeadPosition(); pos != NULL;)
   {
      // returns a CComPtr, so we don't have to release that one at
least
      hr =
m_oracles.GetNext(pos).QueryInterface(&m_adjusters[actualcount]);
      if (hr == S_OK) {
         // m_adjusters is now valid. So notify that we're starting
timing.
        // and ignore the return value for now...although in the future,
we
         // may want to fail start timing if the adjuster is unable to
cope.
         TIMING_PROPERTIES tProps;
         tProps.iter_end = pScript->m_events.end();
         tProps.iter_rend = pScript->m_events.rend();
         tProps.pFilterFunc = m_pFilterFunc;
         hr = m_adjusters[actualcount]->NotifyStartTiming(&tProps);
         hr = m_adjusters[actualcount]->NotifySignalTiming(&tsp);
         actualcount++;
      } else {
         ASSERT(hr = E_NOINTERFACE);
      }
   }
   // size it downwards, as necessary
   m_adjusters.SetCount(actualcount);

   // display onscreen
   PrepareOverlay(pScript, 0);

   // but if it IS paused, then we might want to force an update of the
video
   // by flushing the graph.
```

```
   OAFilterState fs = State_Stopped;
   m_pVideoFrame->m_pMC->GetState(2, &fs);
   if (fs == State_Paused) {
      REFERENCE_TIME rtCur;
      if (m_pVideoFrame->m_pMS && SUCCEEDED(m_pVideoFrame->m_pMS-
>GetCurrentPosition(&rtCur))) {
         hr = m_pVideoFrame->m_pMS->SetPositions(&rtCur,
AM_SEEKING_AbsolutePositioning,
            NULL, AM_SEEKING_NoPositioning);
      }
   }
   // we're live
   m_bTiming = true;
   return true;
} // end StartTiming

// The on-the-fly timing engine.
// This is where the magic happens.
bool CSubTekApp::SignalTiming(UINT nChar, UINT nFlags)
{
   // translate flags to meaningful values
   bool bKeyDown = !(nFlags & KF_UP);     // true if this is a KeyDown;
false if this is a KeyUp
                                    // (KeyDown is more interesting,
usually)
   // not interested in repeat down characters
   if (bKeyDown && (nFlags & KF_REPEAT))
      return false;
   // FIRST, have to confirm that we are in on-the-fly timing mode
   // if not, then ignore.
   if (!m_bTiming)
      return false;
   // also have to confirm that we are timing some media
   if (!m_pVideoFrame || !m_pVideoFrame->IsMediaLoaded()) {
      // what happened? some kind of error. Or, the user unloaded the
media file(s)
      // before turning off the timing engine
      m_bTiming = false;
      m_queEvents.clear();       // 7/20/2005: this also makes nonhot
      return false;
   }

   // grab the state data.
   REFERENCE_TIME rtCur = 0LL;
   // ASSUMPTION: the time format is in 100ns increments,
TIME_FORMAT_MEDIA_TIME
   // TODO: if this fails, then just return or something, because we
have no real
   // basis to grab a time
   VERIFY(SUCCEEDED(m_pVideoFrame->m_pMS->GetCurrentPosition(&rtCur)));

   if (!(bKeyDown ? !(nFlags & KF_REPEAT) : nFlags & KF_REPEAT))  //
bKeyUp IMPLIES that the key was previously held down
   {
```

```
      return false;        // this may happen in odd corner cases, so
rather than ASSERTing, we'll just ignore. It might happen, for example,
      // if a key is pressed right at the beginning of a timing sequence
and is then released, especially if said key is a hotkey
      // for SubTek and is not otherwise relevant to the timing process.
   }

   ASSERT(m_pScriptFrame->GetActiveDocument()-
>IsKindOf(RUNTIME_CLASS(CSubTekScript)));
   CSubTekScript *pScript = (CSubTekScript *)m_pScriptFrame-
>GetActiveDocument();
   // shorthand
   CSTEventList::iterator &ae = pScript->m_activeevent;
   HRESULT hr = S_OK;

   // set up notifier structure
   TIMING_SIGNAL_PROPERTIES tsp = { &m_queEvents, &ae, 0 };

   // events: "h"down, up; "j"down, up; "k"down, up; "l"down; "n"down;
";"down

   // "j"
   if (nChar == m_vkMain) {
      if (bKeyDown) {
         // continuous adjacent event?
         // in that case, the user will press j->k->j
         // also, this may happen if "h" is pressed and the user hands
off to "j"
         // (for adjacent events)
         if (!m_queEvents.empty() &&
         (GetKeyState(m_vkAdjacent) & 0x8000 ||
GetKeyState(m_vkPrevStart) & 0x8000)) {
            // this SHOULD be true; however, the user may be lethargic
or extremely lazy
            // and may just be holding "k" while pressing "j" to set
lots of transition points.
            // but, I *guess* that's acceptable...might even be easier.
            // ASSERT(m_queEvents.size() % 2 == 0);
            NextEvent(ae);
            m_queEvents.push_back(std::make_pair(ae, rtCur));
            // notify the adjusters; ignore if they complain
            for (size_t k = 0; k < m_adjusters.GetCount(); k++) {
               tsp.nAdvance = +1;
               hr = m_adjusters[k]->NotifySignalTiming(&tsp);
            }
            PrepareOverlay(pScript, +1);
         } else {
            // this should be the first time.
            // confirms that this is the first time (no side effects)--
but
            // if the user is merely "reinstating" the timing session,
then we'll just ignore.
            // "j" only has effect if we have an actual event to time
            // otherwise, there is no effect.
```

```
            if (m_queEvents.empty() && ae != pScript->m_events.end()) {
               // add to the event queue
               m_queEvents.push_back(std::make_pair(ae, rtCur));


               // notify the adjusters; ignore if they complain
               for (size_t k = 0; k < m_adjusters.GetCount(); k++) {
                  tsp.nAdvance = 0;
                  hr = m_adjusters[k]->NotifySignalTiming(&tsp);
               }
               // prepare the overlay: output the current text, etc.
               PrepareOverlay(pScript, 0);
            }
         }
      } else {
         // if another key ("k") is down, ignore
         if (!(GetKeyState(m_vkAdjacent) & 0x8000)) {

         // if the queue is empty, ignore (other keys aren't down, but
whatever)
         if (!m_queEvents.empty()) {

            NextEvent(ae); // whenever this moves, the position of the
active should also move.
            // (optional) run preoracleadjustments
            // TODO: run oracleadjustments on the Start, End, Event,
[and current a/v data]
            // TODO: if OracleAdjustments takes a LONG time, it should
be spun off as a separate,
            // low-priority thread.
            // for now, just save the exact start and end time.
            OracleAdjustments(m_queEvents, rtCur, m_adjusters);
               pScript->UpdateAllViews(NULL, 0,
&CSubTekScriptUpdate(m_queEvents.front().first,
                  ++m_queEvents.back().first));
            // (optional) Run post-oracle adjustments predicate
            m_queEvents.clear(); // this also makes nonhot

            for (size_t k = 0; k < m_adjusters.GetCount(); k++) {
               tsp.nAdvance = +1;
               hr = m_adjusters[k]->NotifySignalTiming(&tsp);
            }
            // prepare overlay
            PrepareOverlay(pScript, +1);

         }}
      }
   // "k"
   } else if (nChar == m_vkAdjacent) {
      // NOTE: This is my experience with GetKeyState, by inspection:
      // 0xFF80 == keydown
      // 0x0001 == toggle (keyboard light on)
      // so the "high order bit" can be masked by 0x8000, and the "low-
order
```

```
      // bit" can be masked by 0x0001.
      SHORT keyState = GetKeyState(m_vkMain);
      SHORT keyStateH = GetKeyState(m_vkPrevStart);
      if (bKeyDown) {
         if (keyState & 0x8000 || keyStateH & 0x8000) {
            // "j" or "h" key is down
            ASSERT(m_queEvents.size());
            NextEvent(ae);

            // add to the event queue
            m_queEvents.push_back(std::make_pair(ae, rtCur));
            // notify the adjusters; ignore if they complain
            tsp.nAdvance = +1;
            for (size_t k = 0; k < m_adjusters.GetCount(); k++) {
               hr = m_adjusters[k]->NotifySignalTiming(&tsp);
            }

            // prepare overlay
            PrepareOverlay(pScript, +1);
         } else {
            // "j" key is not down
            // hence, this must be the first time.
            // confirms that this is the first time (no side effects)
            // Thus the queue should be empty, unless the user pressed a
button down, "left" somewhere
            // (i.e., switched windows or focus so that we didn't get a
key up), and is now
            // pressing the button down again. If that's the case, we
ignore.
            // "j" only has effect if we have an actual event to time
            // otherwise, there is no effect.
            if (m_queEvents.empty() && ae != pScript->m_events.end()) {
               // add to the event queue
               m_queEvents.push_back(std::make_pair(ae, rtCur));


               // notify the adjusters; ignore if they complain
               for (size_t k = 0; k < m_adjusters.GetCount(); k++) {
                  tsp.nAdvance = 0;
                  hr = m_adjusters[k]->NotifySignalTiming(&tsp);
               }
               // prepare the overlay: output the current text, etc.
               PrepareOverlay(pScript, 0);
            }
         }
         // don't care about toggle status (0x1)
      } else {
         // key is up
         // if the "j" key is down, don't flush yet. just ignore
         if (!(keyState & 0x8000) && m_queEvents.size()) {
            // there SHOULD be an even number of events in the queue.
            // but there may not be, mainly because the user may have
been lethargic
            // and may have held j->k->jup->->j->
```

```
            // jup (instead of kup) [this up gets ignored]->->kup [now
we're here, and we have
            // an odd number of events, instead of an even #]

            // the following ASSERT will fail if the user tried to skip
forwards or backwards
            // (thus canceling & emptying the queue) while still holding
down k. In that case,
            // we should just ignore--see the stipulation above that the
que has data.
            // ASSERT(m_queEvents.back().first == ae);

            NextEvent(ae);
            OracleAdjustments(m_queEvents, rtCur, m_adjusters);
               pScript->UpdateAllViews(NULL, 0,
&CSubTekScriptUpdate(m_queEvents.front().first,
                  ++m_queEvents.back().first));
            m_queEvents.clear();
            tsp.nAdvance = +1;
            for (size_t k = 0; k < m_adjusters.GetCount(); k++) {
               hr = m_adjusters[k]->NotifySignalTiming(&tsp);
            }

            PrepareOverlay(pScript, +1);
         }
      }
   // "l" (now "n")
   } else if (nChar == m_vkPrevEnd) {
      if (bKeyDown) {
         if (m_queEvents.empty()) {
            // "set previous end time"
            CSTEventList::iterator prevevent = ae;
            if (PrevEvent(prevevent)) {
               std::deque<CSubTekApp::quepair> oneshotque;
               oneshotque.push_back(std::make_pair(prevevent, rtCur));
               TIMING_SIGNAL_PROPERTIES onetsp;
               onetsp.pQue = &oneshotque;
               onetsp.nAdvance = 0;
               onetsp.pActiveEvent = &ae; // this event is still active
               CSTEventList::iterator enditer = prevevent;
               enditer++;
               for (size_t k = 0; k < m_adjusters.GetCount(); k++) {
                  hr = m_adjusters[k]->NotifySignalTiming(&onetsp);
               }
               OracleAdjustments(oneshotque, rtCur, m_adjusters);
               // one event has been updated.
               pScript->UpdateAllViews(NULL, 1,
&CSubTekScriptUpdate(oneshotque.front().first,
                  enditer));
               // restore previous deque as the active deque (hence the
HOT value)
               for (size_t k = 0; k < m_adjusters.GetCount(); k++) {
                  hr = m_adjusters[k]->NotifySignalTiming(&tsp);
               }
```

```
            }
         } else {
            // it has size; hence, "j" or "k" are down (but they don't
actually have to be--
            // what's important is that we are still in the middle of
compiling a queue of
            // events)
            // the meaning here is "fix previous adjacent time"
            CSubTekApp::quepair &hotref = m_queEvents.back();
            hotref.second = rtCur;  // immediately previous end/start
time now replaced with current time
            // should notify adjusters that the TIME has changed.
            tsp.nAdvance = 0;
            for (size_t k = 0; k < m_adjusters.GetCount(); k++) {
               hr = m_adjusters[k]->NotifySignalTiming(&tsp);
            }
         }
      } else {
         // do nothing
      }
   // "h"
   } else if (nChar == m_vkPrevStart) {
      if (bKeyDown) {
         // "redo previous start time"
         if (m_queEvents.empty()) {
            // nothing is pressed (we hope). So, set the active event to
the previous event.
            if (ae == pScript->m_events.begin()) {
               // oops; can't do anything.
            } else {
               PrevEvent(ae);
               // add to the event queue
               m_queEvents.push_back(std::make_pair(ae, rtCur));

               // notify the adjusters; ignore if they complain
               for (size_t k = 0; k < m_adjusters.GetCount(); k++) {
                  tsp.nAdvance = -1;
                  hr = m_adjusters[k]->NotifySignalTiming(&tsp);
               }
            }
         } else {
            // treat as "redo last adjacent time"
            m_queEvents.back().second = rtCur;
            // should notify adjusters that the TIME has changed.
            tsp.nAdvance = 0;
            for (size_t k = 0; k < m_adjusters.GetCount(); k++) {
               hr = m_adjusters[k]->NotifySignalTiming(&tsp);
            }
         }
      } else {
         // h up only has effect when there are events in the que: we
either
         // run oracle adjustments on the que or we
         if (!m_queEvents.empty()) {
```

```
            // if j or k are down, then we hand off to them.
            SHORT keyStateJ = GetKeyState(m_vkMain);
            SHORT keyStateK = GetKeyState(m_vkAdjacent);
            if (keyStateJ & 0x8000 || keyStateK & 0x8000) {
               // do nothing
            } else {
               ASSERT(m_queEvents.size() == 1);
               // adjust the que.
               NextEvent(ae);
               OracleAdjustments(m_queEvents, rtCur, m_adjusters);
               pScript->UpdateAllViews(NULL, 0,
&CSubTekScriptUpdate(m_queEvents.front().first,
                  ++m_queEvents.back().first));
               m_queEvents.clear();
               tsp.nAdvance = +1;
               for (size_t k = 0; k < m_adjusters.GetCount(); k++) {
                  hr = m_adjusters[k]->NotifySignalTiming(&tsp);
               }
            }
         }
      } // end if h is down or up
   // "n" (now "l")
   } else if (nChar == m_vkGotoPrev) {
      if (bKeyDown) {
         // semantics: if event is hot, cancel event and queue
         // so that "j"up, etc. have no effect
         if (m_queEvents.size()) {

            m_queEvents.clear();

            tsp.nAdvance = 0;
            for (size_t k = 0; k < m_adjusters.GetCount(); k++) {
               hr = m_adjusters[k]->NotifySignalTiming(&tsp);
            }

            PrepareOverlay(pScript, 0);
         } else {
         // else, go to previous event and make previous event active
(not hot)
            ASSERT(m_queEvents.empty());  // this should be obvious (so
debug only)
            // don't advance/retreat if we're already at the beginning
            bool atbegin = ae == pScript->m_events.begin();
            PrevEvent(ae);
            tsp.nAdvance = atbegin ? 0 : -1;
            for (size_t k = 0; k < m_adjusters.GetCount(); k++) {
               hr = m_adjusters[k]->NotifySignalTiming(&tsp);
            }

            PrepareOverlay(pScript, -1);
         }
      } else {
         // do nothing
      }
```

```
   // ";"
   } else if (nChar == m_vkGotoNext) {
      if (bKeyDown) {
         // (but don't advance if we're already at the end...except
always clear the queue)
         bool atend = ae == pScript->m_events.end();
         NextEvent(ae);
         if (m_queEvents.size()) {

            m_queEvents.clear();
         }
         tsp.nAdvance = (atend ? 0 : 1);

         for (size_t k = 0; k < m_adjusters.GetCount(); k++) {
            hr = m_adjusters[k]->NotifySignalTiming(&tsp);
         }
      } else {
         // do nothing
      }
   } else if (nChar == m_vkInstantReplay) {
      if (bKeyDown && m_pVideoFrame) {
         m_pVideoFrame->OnPlaybackInstantReplay();
      }
   } // end mega if statement
   // post-cleanup??
   return true;
} // end SignalTiming

// stop the timing engine
void CSubTekApp::StopTiming()
{
   // we're offline
   m_bTiming = false;

   // keep current event active; keep current predicate
   // but empty all queues and set status to non-hot

   m_queEvents.clear();
   HRESULT hr = S_OK;
   for (size_t k = 0; k < m_adjusters.GetCount(); k++) {
      // at this point, just ignore if the adjuster complains
      hr = m_adjusters[k]->NotifyStopTiming();
   }
   // release all adjusters
   m_adjusters.RemoveAll();
} // end Stop Timing Engine
```

Appendix C: Alpha and Beta Tester Agreements

As described in System Evaluation (p. 43), I conducted extensive usability studies with and gathered detailed user feedback from different individuals. While I remain committed to releasing this software under an open-source style license once it reaches a publishable state, I decided to keep the SubTek software and technologies confidential until I decide on the most appropriate license model.

Beta testers agreed to the Confidentiality Agreement and the SubTek Prerelease License Agreement at the same time. Beta testers who also performed alpha testing and development feedback first agreed to the Confidentiality Agreement, and then agreed to the SubTek Prerelease License Agreement. The following texts are the latest versions of the Confidentiality and Prerelease License Agreements.

Confidentiality Agreement

The "recipient" is [Name of Agreeing Party].
During the course of discussions and other events, the recipient may learn of the details or implementations of the subtitling system(s) that Sean Leonard is developing (collectively, "Confidential Information"). The recipient recognizes that the Confidential Information is the property of Sean Leonard and agrees to keep this information confidential, including, but not limited to, taking reasonable steps to prevent disclosure or dissemination to any other party.
Information that is NOT Confidential Information is information that the recipient can demonstrate that he/she had prior to receipt of information from Sean Leonard, information that becomes known to the public through no fault of the recipient, information that becomes known to the recipient from a third party that has a lawful right to disclose the information, and information that was public knowledge before the disclosure of the

information to the recipient. No implied license to the technology or information is to be granted to the recipient based upon this agreement. This agreement shall begin upon the recipient's acceptance of this agreement. This agreement shall terminate for specific Confidential Information when Sean Leonard releases that specific Confidential Information to the public in the form of one or more subtitling systems embodying the specific Confidential Information.

SubTek Prerelease License Agreement

The beta tester is [Name of Agreeing Party].

Sean Leonard grants the beta tester a license to reproduce and use the beta of SubTek(tm)--the System for Rapid Subtitling, and supporting materials, to perform experiments and tests as described herein. The beta tester shall test the software to determine its usefulness and evaluate its features. Beta testing comprises four steps: 1) agreeing to these terms, 2) completing a preliminary survey, 3) receiving and testing SubTek according to instructions given, and 4) completing a post-testing survey; additionally, the beta tester must respond to Sean Leonard's reasonable requests for information pursuant to these four steps.

Hereinafter, Sean Leonard's release of SubTek, parts of SubTek, or related software or systems embodying SubTek technology to the public are collectively "the public release."

Subject to the following conditions, the beta tester may use the software to perform any personal activity--commercial or noncommercial--related to the creation, development, and modification of subtitles, through December 31, 2005 or until the public release, whichever is later (this period is "the prerelease period"). This privilege is valid: 1) if and only if the beta tester agrees to notify Sean Leonard at least once a month with meaningful feedback on the software's usefulness, on desired features, on outstanding issues, and on works created or modified with the software (where the disclosure of works is not barred by other confidentiality agreements); and 2) after and only after the beta tester completes beta testing.

The beta tester may not reverse engineer or otherwise disassemble the software, but the beta tester may provide a "crash dump" to Sean Leonard in case of unintentional software malfunction, and may inspect the local memory area of interest pursuant to identifying the malfunction.

After the prerelease period, the beta tester must cease use of this beta software and delete all outstanding copies. This software is not published, sold, traded, or offered for sale, and this software comes WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. This license is not transferable or divisible. Sean Leonard retains all intellectual property rights related to

SubTek. The beta tester will take reasonable precautions, as agreed to in the CONFIDENTIALITY AGREEMENT, to treat the SubTek software and supporting materials as confidential. This license agreement shall begin upon the beta tester's acceptance of this license agreement.
This license agreement does not extend to the public release, so the beta tester may use those future public releases as any other member of the public. This license agreement extends to any future beta created during the prerelease period. The beta tester may request, and Sean Leonard will attempt to reasonably provide on request, any future beta versions that fall under this extension.